Galileo Computing

Apps programmieren für iPhone und iPad

Inkl. Xcode, Debugging, Versionierung, zahlreiche Praxisbeispiele. Aktuell zu iOS 6

Bearbeitet von
Klaus M. Rodewig, Clemens Wagner

Neuausgabe 2012. Buch. 1012 S.
ISBN 978 3 8362 1915 0
Format (B x L): 168 x 24 cm

Weitere Fachgebiete > EDV | Informatik > Programmiersprachen: Methoden > PDA & Handheld Programmierung

schnell und portofrei erhältlich bei

beck-shop.de
DIE FACHBUCHHANDLUNG

Die Online-Fachbuchhandlung beck-shop.de ist spezialisiert auf Fachbücher, insbesondere Recht, Steuern und Wirtschaft. Im Sortiment finden Sie alle Medien (Bücher, Zeitschriften, CDs, eBooks, etc.) aller Verlage. Ergänzt wird das Programm durch Services wie Neuerscheinungsdienst oder Zusammenstellungen von Büchern zu Sonderpreisen. Der Shop führt mehr als 8 Millionen Produkte.

Klaus M. Rodewig, Clemens Wagner

Apps programmieren für iPhone und iPad

Das umfassende Handbuch

Galileo Press

Auf einen Blick

1 Hello iPhone 21

2 Grundlagen 55

3 Views und Viewcontroller 199

4 Alles unter Kontrolle 307

5 Daten, Tabellen und Controller 405

6 Models, Layer, Animationen 529

7 Programmieren, aber sicher 655

8 Datenserialisierung und Internetzugriff 765

9 Jahrmarkt der Nützlichkeiten 883

10 Sicherer Entwicklungszyklus 975

Inhalt

Geleitwort 15
Vorwort 17

1 Hello iPhone 21

1.1 Voraussetzungen 22
1.1.1 Das SDK und die Entwicklerprogramme 22
1.1.2 Der Simulator 25
1.1.3 Test der Arbeitsumgebung 26

1.2 App geht's 32
1.2.1 Ein neues Projekt 32
1.2.2 Sehr verbunden 37
1.2.3 Netzwerk und Ausgabe 41
1.2.4 Test auf einem Gerät 43
1.2.5 Entwicklungszertifikat und Entwicklungsprofile 44
1.2.6 Entwicklerprofile ohne Wildcards 50
1.2.7 Profilprobleme 52

1.3 Zusammenfassung 54

2 Grundlagen 55

2.1 Objektorientierte Programmierung 56
2.1.1 Objekte und Abstraktion 58
2.1.2 Vererbung 61
2.1.3 Überschreiben von Methoden und spätes Binden 64
2.1.4 Objektorientierung in Objective-C 65
2.1.5 Die Architektur von iOS-Programmen 73

2.2 Hefte raus, Klassenarbeit! 75
2.2.1 Controller und View in der Praxis 77
2.2.2 Modellbau 85
2.2.3 Initializer und Methoden 90
2.2.4 Vererbung 100
2.2.5 Kategorien 112
2.2.6 Protokolle 117

2.2.7 Vorwärtsdeklarationen ... 118
2.2.8 Kommunikation zwischen den Schichten ... 119
2.2.9 Delegation ... 125

2.3 Speicherverwaltung, Propertys und Key-Value-Coding ... 135
2.3.1 Der Stack ... 136
2.3.2 Der Heap ... 136
2.3.3 Referenzen zählen ... 139
2.3.4 Die Speicherverwaltungsregeln ... 143
2.3.5 Autorelease-Pools ... 149
2.3.6 Propertys und Accessoren ... 153
2.3.7 Key-Value-Coding ... 158
2.3.8 Abschließende Bemerkungen ... 161

2.4 Automatisches Referenzenzählen ... 162
2.4.1 Zurück in die Zukunft ... 163
2.4.2 Weakie und die starken Zeiger ... 164
2.4.3 Autorelease-Pools ... 168
2.4.4 Einzelgänger ... 168
2.4.5 Migration bestehender Projekte ... 170

2.5 Das Foundation-Framework ... 173
2.5.1 Mutables und Immutables ... 174
2.5.2 Elementare Klassen ... 177
2.5.3 Collections ... 184

2.6 Blöcke ... 189
2.6.1 Rückruffunktionen ... 193

2.7 Namenskonventionen ... 196

2.8 Zusammenfassung ... 197

3 Views und Viewcontroller 199

3.1 Eigene View-Klassen in Cocoa Touch ... 199
3.1.1 Zeichnen in Cocoa Touch ... 208
3.1.2 Zeitberechnung ... 217
3.1.3 View-Erzeugung über NIB-Dateien ... 220
3.1.4 Aktualisierung der Zeitanzeige ... 222
3.1.5 Wiederverwendbarkeit von Views ... 225

3.2 Views und Viewcontroller ... 226
3.2.1 Outlets ... 228
3.2.2 Outlet-Collections ... 231

3.2.3 Containerviews ... 233
3.2.4 Viewhierarchien ... 234
3.2.5 Actions ... 235
3.2.6 Ereignisse ... 238
3.2.7 Controlzustände und Buttons ... 242
3.2.8 Direkte Gestenverarbeitung ... 246
3.2.9 Übergänge ... 259
3.2.10 Der Lebenszyklus eines Viewcontrollers ... 261
3.2.11 Speicher- und Ressourcenverwaltung des Viewcontrollers ... 266

3.3 Lokale Benachrichtigungen ... 268
3.3.1 Benachrichtigungen versenden ... 270
3.3.2 Benachrichtigungen verarbeiten ... 273

3.4 Eine App für alle ... 278
3.4.1 Das Retina-Display ... 280
3.4.2 Sprachkursus für die App ... 282
3.4.3 Es funktioniert nicht ... 286
3.4.4 Launch-Images und iPhone 5 ... 287
3.4.5 Universelle Apps ... 290

3.5 Fehlersuche ... 292
3.5.1 Logging ... 293
3.5.2 Der Debugger ... 294
3.5.3 Die Debuggerkonsole ... 298
3.5.4 Laufzeitfehler ... 304

4 Alles unter Kontrolle 307

4.1 Viewcontroller, XIBs und Storyboards ... 307
4.1.1 Erstellung von Viewcontrollern über XIB-Dateien ... 308
4.1.2 Applikationsinitialisierung ... 311
4.1.3 Storyboards ... 316
4.1.4 Modale Dialoge ... 324
4.1.5 Pop-over ... 329
4.1.6 Übergänge ... 332

4.2 Der Navigationcontroller ... 333
4.2.1 Viewcontroller anzeigen und entfernen ... 334
4.2.2 Die Navigationsleiste ... 335
4.2.3 Der Zurückbutton ... 338
4.2.4 Gehe drei Felder zurück ... 339

4.2.5 Die Werkzeugleiste ... 343
4.2.6 Navigations- und Werkzeugleisten gestalten ... 345

4.3 Navigation- und Pop-over-Controller in der Praxis ... 347
4.3.1 Navigationcontroller anlegen ... 351
4.3.2 Einen Dialog für die Einstellungen gestalten ... 352
4.3.3 Einstellungen dauerhaft speichern ... 354
4.3.4 Anpassung an das iPad ... 359

4.4 Der Splitviewcontroller ... 362
4.4.1 Das Splitviewcontroller-Delegate ... 363
4.4.2 Ein praktisches Beispiel ... 365
4.4.3 Anpassung des Rotationsverhaltens ... 367

4.5 Der Tabbarcontroller ... 371
4.5.1 Aufbau einer Reiternavigation ... 372
4.5.2 Plaketten ... 374
4.5.3 Für ein paar Controller mehr ... 374
4.5.4 Der Tabbarcontroller im Storyboard ... 379
4.5.5 Die Gestaltungsmöglichkeiten für die Tabbar unter iOS 5 ... 380

4.6 Der Pageviewcontroller ... 381
4.6.1 Einen Pageviewcontroller erzeugen ... 382
4.6.2 Die Datenquelle und das Delegate ... 384

4.7 Eigene Container- und Subviewcontroller ... 386
4.7.1 Container- und Subviewcontroller ... 386
4.7.2 Eigene Subviewcontroller ... 391
4.7.3 Eigene Delegate-Protokolle erstellen ... 396
4.7.4 Delegate-Protokolle erweitern ... 397

4.8 Regelbasierte Anpassung der Oberfläche ... 399
4.8.1 Gestaltungsregeln 399
4.8.2 ... und Gestaltungsmöglichkeiten ... 402

5 Daten, Tabellen und Controller 405

5.1 Benachrichtigungen ... 405
5.1.1 Benachrichtigungen empfangen ... 406
5.1.2 Eigene Benachrichtigungen verschicken ... 408

5.2 Layoutanpassungen und Viewrotationen ... 410
5.2.1 Flexible Views dank Autosizing ... 410
5.2.2 Lang lebe das Rotationsprinzip! ... 413
5.2.3 Anpassung des Layouts ... 416

5.3 Autolayout ... 419
5.3.1 Restriktionen im Interface Builder festlegen ... 419
5.3.2 Restriktionen per Programmcode erstellen ... 425
5.3.3 Autolayout und Lokalisierung ... 431

5.4 Core Data ... 434
5.4.1 Datenmodellierung ... 435
5.4.2 Implementierung von Entitätstypen ... 438
5.4.3 Einbindung von Core Data ... 440
5.4.4 Der Objektkontext ... 441
5.4.5 Die Nachrichten des Objektkontexts ... 443
5.4.6 Anlegen und Ändern von Entitäten in der Praxis ... 443

5.5 Texte, Bilder und Töne verwalten ... 447
5.5.1 Die Tastatur betritt die Bühne ... 447
5.5.2 Fotos aufnehmen ... 453
5.5.3 Töne aufnehmen und abspielen ... 461

5.6 Tableviews und Core Data ... 472
5.6.1 Tableviews ... 472
5.6.2 Tabellenzellen gestalten ... 475
5.6.3 Tableviews unter iOS 5 ... 478
5.6.4 Buttons in Tabellenzellen ... 480
5.6.5 Zellen löschen ... 481

5.7 Core Data II: Die Rückkehr der Objekte ... 482
5.7.1 Prädikate ... 483
5.7.2 Aktualisierung des Tableviews ... 486
5.7.3 Das Delegate des Fetchedresultscontrollers ... 489
5.7.4 Tabelleneinträge suchen ... 493

5.8 Twitter ... 496
5.8.1 Der Twitter-Account ... 497
5.8.2 Hinzufügen des Twitter-Frameworks ... 498
5.8.3 Integration in das Fototagebuch ... 500

5.9 Collectionviews ... 507
5.9.1 Der Collectionviewcontroller ... 508
5.9.2 Gitterdarstellung ... 509
5.9.3 Zellen und die Datenquelle ... 512
5.9.4 Ergänzende Views ... 514
5.9.5 Freie Layouts ... 516
5.9.6 Freie Layouts und ergänzende Views ... 523
5.9.7 Zellen auswählen ... 525
5.9.8 Decorationviews ... 526

6 Models, Layer, Animationen 529

6.1 Modell und Controller ... 530
6.1.1 iOS Next Topmodel ... 530
6.1.2 View an Controller ... 535
6.1.3 Modell an Controller ... 541
6.1.4 Undo und Redo ... 543
6.1.5 Unit-Tests ... 548

6.2 Als die Views das Laufen lernten ... 556
6.2.1 Animationen mit Blöcken ... 558
6.2.2 Transitionen ... 565
6.2.3 Zur Animation? Bitte jeder nur einen Block! ... 568

6.3 Core Animation ... 570
6.3.1 Layer ... 570
6.3.2 Vordefinierte Layer-Klassen ... 577
6.3.3 Unser Button soll schöner werden ... 580
6.3.4 Spieglein, Spieglein an der Wand ... 582
6.3.5 Der bewegte Layer ... 591
6.3.6 Daumenkino ... 595
6.3.7 Relativitätstheorie ... 598
6.3.8 Der View, der Layer, seine Animation und ihr Liebhaber ... 602
6.3.9 Die 3. Dimension ... 608

6.4 Scrollviews und gekachelte Layer ... 611
6.4.1 Scrollen und Zoomen ... 612
6.4.2 Die Eventverarbeitung ... 616
6.4.3 Scharfe Kurven ... 622
6.4.4 Ganz großes Kino ... 625
6.4.5 PDF-Dateien anzeigen ... 630

6.5 Über diese Brücke musst du gehen ... 637
6.5.1 Toll-Free Bridging und ARC ... 637
6.5.2 C-Frameworks und ARC ... 639

6.6 Was Sie schon immer über Instruments wissen wollten, aber nie zu fragen wagten ... 640
6.6.1 Spiel mir das Lied vom Leak ... 642
6.6.2 Ich folgte einem Zombie ... 647
6.6.3 Time Bandits ... 650

7 Programmieren, aber sicher 655

7.1 iOS und Hardware ... 656

7.2 Bedrohungen, Angriffe, Sicherheitslücken und Maßnahmen ... 661

7.2.1 Arten von Sicherheitslücken ... 662

7.3 Threat Modeling ... 664

7.3.1 Erstellen eines Datenflussdiagramms ... 666

7.3.2 STRIDE ... 689

7.3.3 Generische Design-Grundsätze ... 697

7.3.4 Threat Modeling aus der Tube – das Microsoft SDL Threat Modeling Tool ... 698

7.4 Sichere Programmierung in der Praxis ... 703

7.4.1 Authentisierung ... 704

7.4.2 Keychain ... 718

7.4.3 Jailbreak-Erkennung ... 726

7.4.4 Verzeichnisse und Dateiattribute ... 732

7.4.5 Event-Handling ... 735

7.4.6 Screenshots ... 738

7.4.7 Sorgfältiger Umgang mit der Bildschirmsperre ... 739

7.4.8 Struktur und Ordnung im Sandkasten ... 740

7.5 iCloud ... 744

7.5.1 Denkanstöße ... 746

7.5.2 iCloud in der Praxis ... 748

7.5.3 Key Value Storage ... 754

7.5.4 Verschlüsselung in der Cloud ... 756

8 Datenserialisierung und Internetzugriff 765

8.1 Ich packe meine Texte ... 765

8.1.1 Serialisierung von Zeichenketten ... 765

8.1.2 Zeichenkodierungen ... 766

8.1.3 Unicode und UTF-8 ... 767

8.1.4 Die Zeichenkodierung erkennen ... 768

8.1.5 Zeichenketten konvertieren ... 769

8.2 JSON und die URLonauten ... 770

8.2.1 Das JSON-Format ... 771

8.2.2 Einfacher YouTube-Zugriff mit JSON ... 772

8.2.3 URLs erstellen ... 779

8.2.4 JSON-Dokumente schreiben ... 783
8.2.5 Verwendung des JSONKits als Parser ... 784

8.3 XML ... 786
8.3.1 XML in Kürze ... 786
8.3.2 Property-Listen ... 791
8.3.3 SAX ... 795
8.3.4 DOM und XPath ... 810
8.3.5 Der Tag der Entscheidung ... 819

8.4 Daten, Daten, ihr müsst wandern ... 820
8.4.1 Synchrone Kommunikation ... 821
8.4.2 Komplexe Anfragen ... 823
8.4.3 Auf dem Webserver nichts Neues ... 827
8.4.4 Asynchrone Kommunikation ... 833
8.4.5 Große Datenmengen der Übertragung ... 838
8.4.6 Passwortabfragen ... 845
8.4.7 Hier geht die POST ab ... 850
8.4.8 Datei-Upload ... 854
8.4.9 Überprüfung der Erreichbarkeit ... 864

8.5 Karten ... 869
8.5.1 Karten darstellen ... 869
8.5.2 Koordinatensysteme ... 871
8.5.3 Geokoordinaten bestimmen ... 873
8.5.4 Eigene Kartenbeschriftungen ... 875
8.5.5 Routen ... 878

9 Jahrmarkt der Nützlichkeiten 883

9.1 Die Struktur von Xcode 4 ... 883
9.1.1 Das Hauptfenster und die Werkzeugleiste ... 884
9.1.2 Die Navigatorspalte ... 885
9.1.3 Der Utilitybereich ... 887
9.1.4 Der Editor ... 889
9.1.5 Autovervollständigung und Codeschnipsel ... 890
9.1.6 Tastaturkurzbefehle ... 894
9.1.7 Tabs und Fenster ... 895
9.1.8 Liebling, ich habe das SDK geschrumpft ... 896
9.1.9 Snapshots ... 898

9.2 Refactoring ... 901
9.2.1 Refactorings in Xcode ... 901
9.2.2 Methoden auslagern ... 904
9.2.3 Oberklassen erzeugen und Methoden verschieben ... 906
9.2.4 Attribute kapseln und verschieben ... 909

9.3 Der Organizer ... 910
9.3.1 Die Geräteverwaltung ... 910
9.3.2 Projektverwaltung ... 912
9.3.3 Archive und die Archivverwaltung ... 913
9.3.4 Dokumentation ... 916

9.4 Das Buildsystem ... 919
9.4.1 Workspaces, Projekte und Targets ... 920
9.4.2 Klassen in Bibliotheken auslagern ... 921
9.4.3 Bibliotheken wiederverwenden ... 927
9.4.4 Konfigurationen ... 930
9.4.5 Targets ... 934
9.4.6 Schemata ... 941

9.5 Ad-Hoc-Distributionen ... 946
9.5.1 Geräteregistrierung ... 946
9.5.2 Installation über das iPhone-Konfigurationsprogramm ... 949
9.5.3 Ad-Hoc-Distributionen über einen Webserver ... 950

9.6 Versionsverwaltung mit Git ... 952
9.6.1 Lokale Git-Repositorys ... 954
9.6.2 Arbeiten mit verteilten Git-Repositorys ... 955
9.6.3 Git-Integration in Xcode ... 956
9.6.4 Einbindung existierender Git-Repositorys ... 960
9.6.5 GitX ... 962
9.6.6 SourceTree ... 963
9.6.7 Das Kommandozeilenprogramm von Git ... 965
9.6.8 Der eigene Git-Server ... 967

A Sicherer Entwicklungszyklus ... 975

Index ... 993

Kapitel 3
Views und Viewcontroller

»Never put off till tomorrow what you can do the day after tomorrow.«
– Mark Twain

In diesem Kapitel werden Sie einen analogen Wecker programmieren und dabei den grundlegenden Aufbau von iOS-Apps kennenlernen. Dabei stehen die Arbeit mit Xcode und die praktische Anwendung der theoretischen Grundlagen im Vordergrund, die wir in den ersten beiden Kapiteln beschrieben haben. Außerdem gestalten Sie eine einfache eigene Oberfläche, die sowohl eigene Grafiken auf den Bildschirm zeichnet als auch Nutzereingaben verarbeitet. Denn das wichtigste Merkmal eines analogen Weckers ist schließlich das Ziffernblatt mit den Zeigern.

Projektinformation

Den Quellcode des folgenden Beispiels finden Sie auf der DVD unter *Code/Apps/AlarmClock* oder im Github-Repository zum Buch im Unterverzeichnis *https://github.com/Cocoaneheads/iPhone/tree/Auflage_2/Apps/AlarmClock*. Außerdem finden Sie einige der besprochenen Varianten des Projekts unter *Code/Apps/Clock* beziehungsweise *https://github.com/Cocoaneheads/iPhone/tree/Auflage_2/Apps/Clock*.

3.1 Eigene View-Klassen in Cocoa Touch

Und los geht's: Legen Sie ein neues iPhone-Projekt in Xcode aus der Vorlage Single View Application an (siehe Abbildung 3.1), und geben Sie dem Projekt den Namen *AlarmClock*.

Im zweiten Schritt schalten Sie alle Checkboxen aus. Der Code in diesem Kapitel verwendet manuelles Referenzenzählen, um die Speicherverwaltungsregeln zu vertiefen. Das Beispielprojekt basiert außerdem auf XIB-Dateien, da Sie Storyboards erst ab iOS 5 einsetzen können. Sie können aber fast alle Beschreibungen für XIB-Dateien auch auf Storyboards übertragen. Als Produktnamen und Präfix für die Klassennamen geben Sie »AlarmClock« ein (siehe Abbildung 3.2). Danach speichern Sie das Projekt in einem Verzeichnis Ihrer Wahl ab.

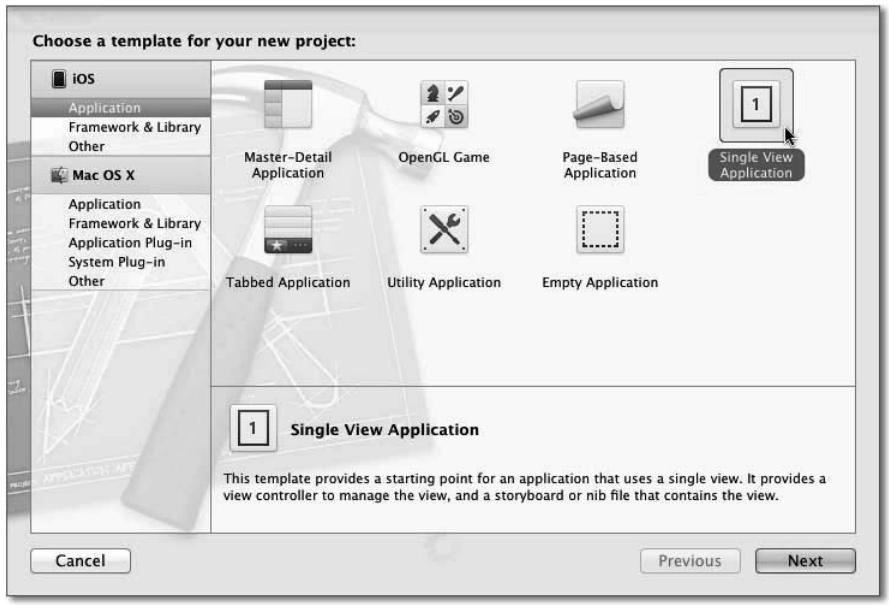

Abbildung 3.1 Auswahl der Projektvorlage für den Wecker

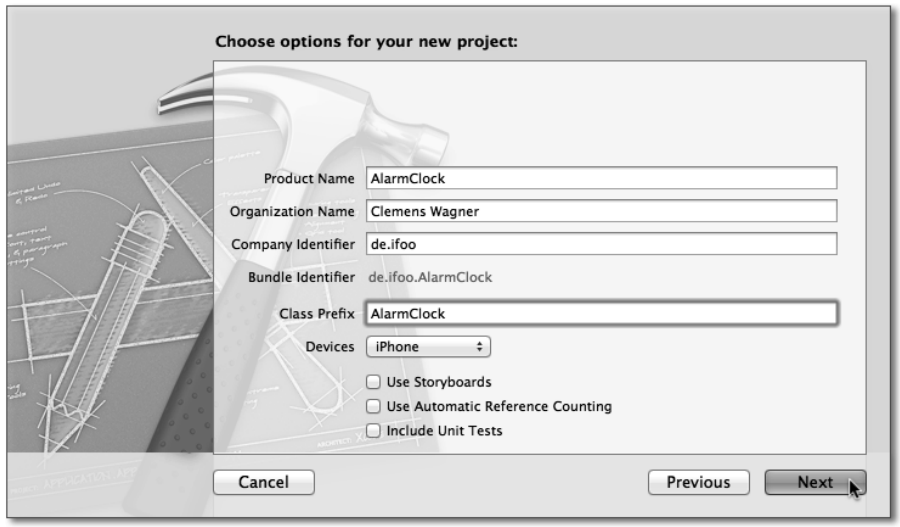

Abbildung 3.2 Die Parameter für das neue Projekt

Als Erstes soll der Wecker ein Ziffernblatt erhalten. Da es dafür keinen fertigen View in Cocoa Touch gibt, müssen Sie eine eigene Viewklasse erstellen. Zum Anlegen der Klasse klicken Sie mit der rechten Maustaste auf die Gruppe AlarmClock und wählen den Punkt New File... aus (siehe Abbildung 3.3).

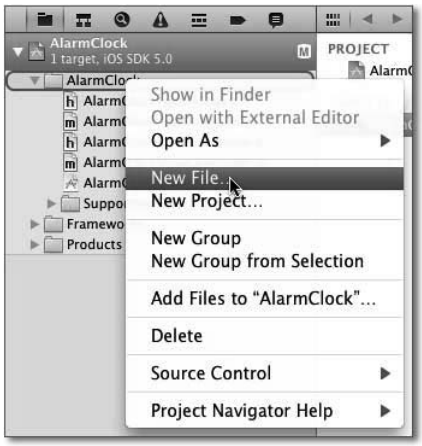

Abbildung 3.3 Hinzufügen einer neuen Klasse

Wählen Sie in der linken Spalte den Punkt Cocoa Touch in der Rubrik iOS aus. Auf der rechten Seite erscheint eine Liste mit Dateivorlagen, aus der Sie Objective-C class auswählen. Geben Sie im nächsten Schritt der Klasse den Namen `ClockView`, und wählen Sie als Superklasse `UIView` aus (siehe Abbildung 3.4) – Xcode erstellt die Klassen dann als Subklasse von `UIView`.

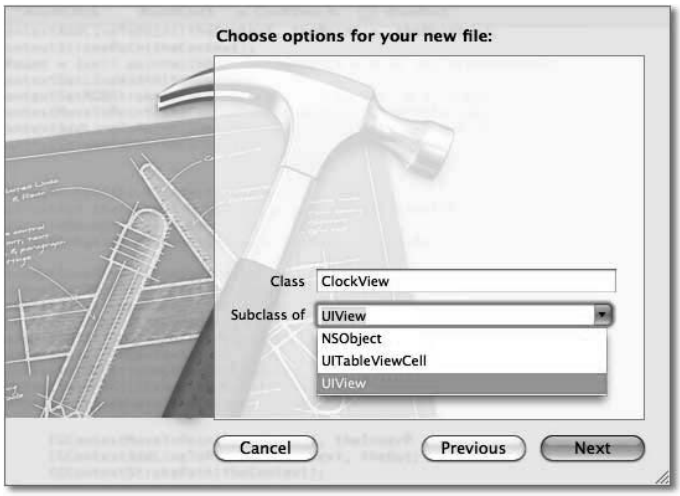

Abbildung 3.4 Subklasse von »UIView«

Im letzten Schritt wählen Sie den Ordner, die Gruppe und das Target für die Header- und die Implementierungsdatei aus (siehe Abbildung 3.5). Hier können Sie einfach die Voreinstellungen übernehmen.

Gruppen und Targets

Dabei ist eine *Gruppe* ist jeweils eine Hierarchieebene in der Dateiansicht von Xcode. Obwohl Gruppen ein gelbes Ordnersymbol als Icon haben, entsprechen sie keinem Dateiordner auf der Festplatte. Mit einem Xcode-Projekt können Sie unterschiedliche Produkte (z. B. Programme) erzeugen. Wie Xcode dieses Produkt erstellt, beschreibt dabei ein *Target*.

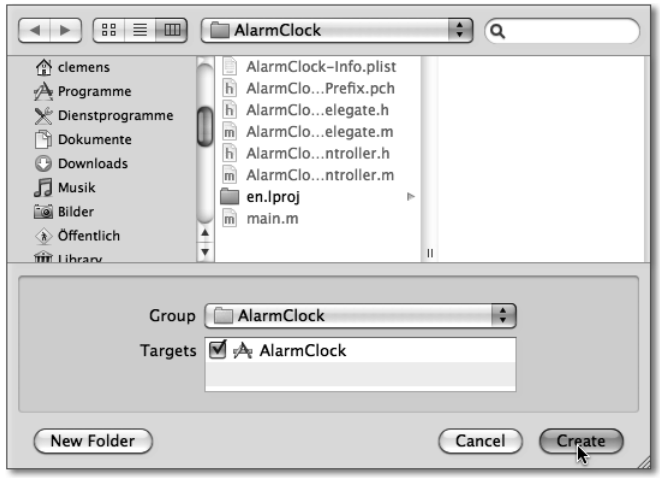

Abbildung 3.5 Anlegen der neuen Klasse in der Gruppe »AlarmClock«

Nach dem Anlegen der Dateien zeigt ein Blick in den Quelltext der Headerdatei der neuen Klasse, dass Xcode eine Subklasse von `UIView` erstellt hat:

```
#import <UIKit/UIKit.h>

@interface ClockView : UIView

@end
```

Listing 3.1 Header-Datei der Klasse »ClockView«

Bevor Sie jedoch die Zeichenlogik für das Ziffernblatt implementieren, erstellen Sie die Oberfläche der App. Dann können Sie das Programm immer wieder starten und sich so die Auswirkungen Ihrer Änderungen ansehen.

Um die neue Viewklasse benutzen zu können, müssen Sie einen View in der XIB-Datei des Viewcontrollers anlegen, die Xcode beim Anlegen des Projekts erzeugt hat. Wählen Sie im Projektnavigator die Datei *AlarmClockViewController.xib* aus. Xcode öffnet dann den Interface Builder und zeigt den enthaltenen View an. Um das Ziffern-

blatt in diesen View einzufügen, ziehen Sie einen `UIView` aus der Objektbibliothek auf den bereits vorhandenen View (siehe Abbildung 3.6).

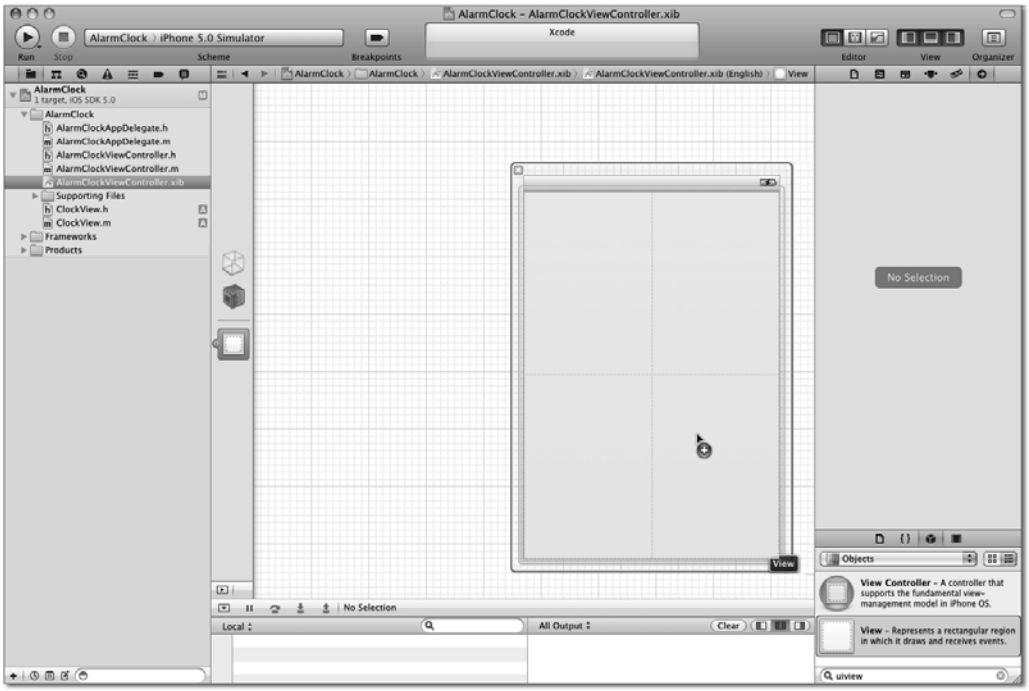

Abbildung 3.6 Einen neuen View im Interface Builder anlegen

Der neue View hat automatisch dieselbe Größe wie der bereits vorhandene erhalten. Da das Ziffernblatt des Weckers aber nicht den ganzen Bildschirm ausfüllen soll, markieren Sie den gerade hinzugefügten View, indem Sie ihn anklicken, und öffnen rechts oben im Xcode-Fenster den Größeninspektor (siehe den Mauszeiger in Abbildung 3.7; alternativ drücken Sie alt+cmd+5).

Der Größeninspektor

In Xcode bis Version 4.4 einschließlich zeigt der Größeninspektor unterhalb der Größeneinstellungen das *Autosizing* anstatt der Autolayouteinstellungen an. In Xcode 4.5 können Sie zwischen Autosizing und Autolayouts im Dateiinspektor der XIB-Datei wechseln; diese Einstellung ist allerdings beim AlarmClock-Beispielprojekt noch irrelevant. Mehr zu diesem Thema finden Sie in Kapitel 5, »Daten, Tabellen und Controller«.

Setzen Sie die Koordinaten und die Größe des Views so, wie in Abbildung 3.7 dargestellt. Er sollte eine quadratische Grundfläche von 320 × 320 Punkten haben und am

Koordinatenursprung (0, 0) liegen. Der Ursprung eines Views ist immer die linke obere Ecke. Die horizontale Ausdehnung wächst nach rechts und die vertikale nach unten.

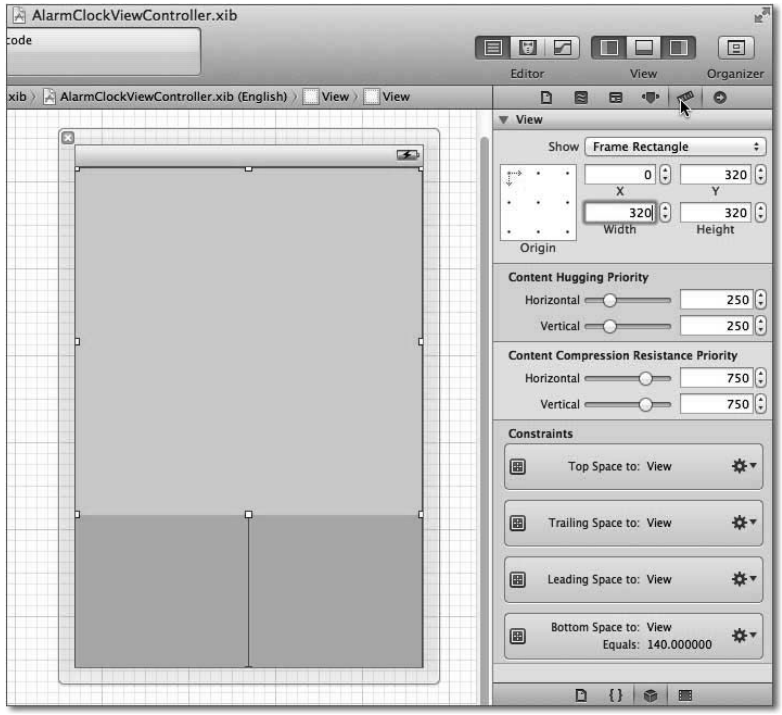

Abbildung 3.7 Der Größeninspektor des Views mit Autolayout

Das Ziffernblatt belegt mit dieser Einstellung nur circa zwei Drittel der oberen Fläche des iPhone-Bildschirms. Sie können das überprüfen, indem Sie das Projekt ausführen. Der Simulator zeigt jetzt eine weiße und darunter eine etwa halb so große graue Fläche an.

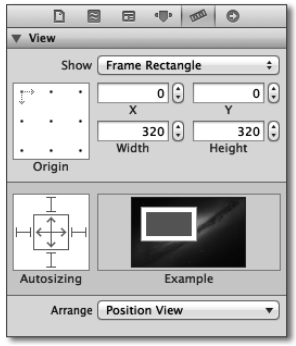

Abbildung 3.8 Der Größeninspektor mit Autosizing

Über das dritte Symbol von links (neben dem Größeninspektor) oder mit alt +cmd+3 wählen Sie den Identitätsinspektor aus. Legen Sie dort über das oberste Eingabefeld, CLASS, die Klasse des Views fest. Sie können den Klassennamen dort entweder direkt eingeben oder aus einer Liste auswählen (siehe Abbildung 3.9). Weisen Sie dem View die neu angelegte Klasse `ClockView` zu.

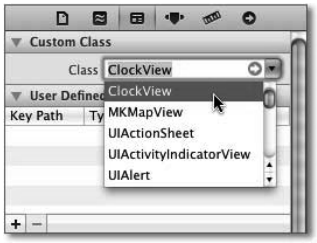

Abbildung 3.9 Der View bekommt die Klasse »ClockView« zugewiesen.

Der Interface Builder kennt die Klassenhierarchie

Beim Erzeugen des Views haben Sie einen UIView verwendet. Damit haben Sie bereits festgelegt, dass als Klasse für diesen View nur `UIView` oder eine ihrer Unterklassen infrage kommt. Daher erscheinen nur genau diese Klassennamen in der Liste. Pfiffigerweise haben Sie ja die designierte Klasse `ClockView` als Unterklasse von `UIView` deklariert, und so kommt sie in der Liste vor, und alles passt.

Die Implementierungsdatei der Klasse `ClockView` enthält bereits zwei Methoden. Die erste Methode heißt `initWithFrame:` und hat einen Parameter vom Typ `CGRect`. Das ist ein zusammengesetzter Datentyp[1] und keine Objective-C-Klasse, der ein Rechteck über eine Koordinate und Seitenlängen (die im Folgenden auch »Größe« genannt werden) darstellt.

Dieser Typ hat Klasse

Sie können einfache Datentypen und C-Strukturen in Cocoa nicht am Namen von Klassen unterscheiden. Zum Beispiel gibt es eine Klasse `NSNumber` und einen einfachen Datentyp `NSInteger`. Es gibt darauf jedoch keinen Hinweis im Namen.

Objective-C erlaubt nur Zeiger als Verweise auf Objekte, weswegen die Deklaration `NSNumber theNumber;`[2] zu der Fehlermeldung

```
interface type cannot be statically allocated
```

führt. Umgekehrt dürfen Sie dagegen Zeiger auf einfache Datentypen deklarieren.

1 Ein C-Struct

2 In der Deklaration fehlt der Stern zwischen dem Klassennamen und der Variable.

Die Deklaration `NSInteger *theInteger;` ist erlaubt, und erst bei der Zuweisung erhalten Sie in der Regel eine Warnung. Falls Sie sich nicht sicher sind, was ein Name im Quelltext beschreibt, können Sie durch cmd + linke Maustaste zur Deklaration dieses Namens gelangen. Das funktioniert praktischerweise mit fast allen Bezeichnern im Quelltext. Bei Bezeichnern aus Cocoa können Sie auch alt + linke Maustaste drücken, um eine kurze Dokumentation zu erhalten.

In Objective-C gibt es keine Konstruktoren, die Sie vielleicht aus anderen Programmiersprachen kennen. Die Objekterzeugung ist vielmehr in Speicherplatzreservierung und Objektinitialisierung aufgeteilt. Die Klassenmethode `alloc` zur Speicherplatzreservierung haben Sie bereits kennengelernt. Sie reserviert den notwendigen Speicherplatz und füllt ihn mit Nullen.

Initialisierungsmethoden kennen Sie ebenfalls schon seit dem zweiten Kapitel. In Objective-C heißen sie per Konvention entweder `init` oder haben das Präfix `init`. Wenn Sie einen View über den Programmcode anlegen, sollten Sie in der Regel die Initialisierungsmethode `initWithFrame:` verwenden. Der Parameter ist ein Rechteck, das die Größe und Position des neuen Views relativ zu dessen *umgebendem View*, dem *Superview*, angibt. Da Cocoa Touch die Views in einer Hierarchie anordnet, hat fast jeder View einen umgebenden View; das ist jeweils dessen Vorgänger in der Hierarchie. Der Aufbau der Methode ist im Allgemeinen wie folgt:

```
- (id)initWithFrame:(CGRect)inFrame {
    self = [super initWithFrame:inFrame];
    if(self != nil) {
        // Initialisierung des Objekts
    }
    return self;
}
```

Listing 3.2 Initialisierungsmethode eines Views

Viewgeometrie

Die Klasse `UIView` besitzt zwei Rechtecke, die die Position und die Ausmaße des Views beschreiben. Die Property `frame` enthält, wie bereits beschrieben, die Position und die Größe des Views relativ zu seinem umgebenden View. Die Position des Frames gibt also die Entfernung der oberen linken Ecke des Views von der oberen linken Ecke seines umgebenden Views an (siehe Abbildung 3.10, links).

Der gezeichnete View muss hingegen nicht auf dem Rechteck liegen, das der *Frame* beschreibt. Diese Koordinaten gibt das Rechteck in der Property `bounds` an.

Die *Bounds* beschreiben also die Lage des Views relativ zu dessen Frame. In den meisten Fällen ist die Position der Bounds der Nullpunkt, und die Größe entspricht der Größe des Frames (Abbildung 3.10, rechts).

Das klingt recht kompliziert. Für den Anfang können Sie sich zwei einfache Faustregeln merken:

1. Sie verändern die Größe und die Position eines Views in der Regel über dessen Frame.
2. Wenn Sie die Koordinaten eines Views relativ zu dessen umgebendem View berechnen möchten, verwenden Sie die Bounds des umgebenden Views, um den Frame des enthaltenen Views zu berechnen.

Das wird später noch an ein paar Beispielen erläutert.

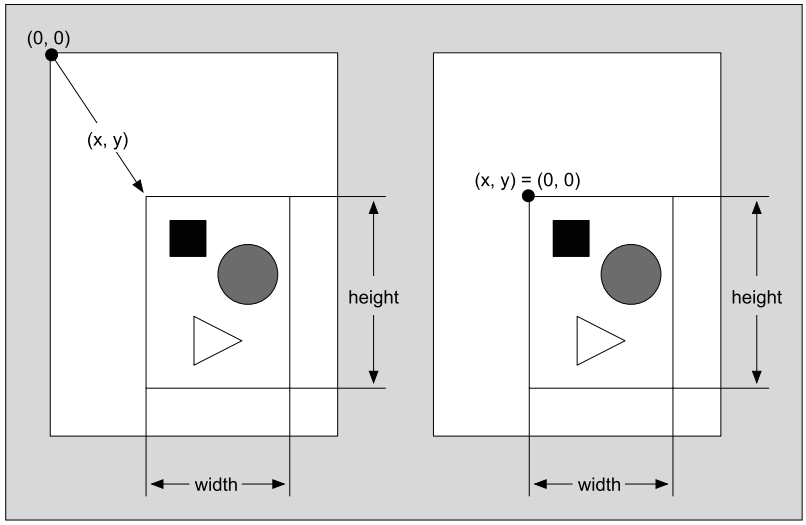

Abbildung 3.10 Rechteck für Frame und Bounds

Die Methode in Listing 3.2 ruft als Erstes die Initialisierungsmethode der Oberklasse auf und weist deren Ergebnis der Variablen `self` zu. Danach überprüft sie, ob `self` nicht `nil` ist. Wenn `self` auf ein Objekt zeigt, kann sie das Objekt initialisieren und schließlich auch zurückgeben.

Vielleicht überrascht es Sie, dass die Initialisierungsmethode `self` verändern kann oder `nil` zurückliefern darf. Das ist indes keine theoretische Möglichkeit, sondern wird auch eingesetzt. Eine Initialisierungsmethode sollte beispielsweise `nil` zurückgeben, wenn sie das Objekt nicht initialisieren kann. Die Initialisierungsmethode `initWithString:` der Klasse `NSURL` gibt beispielsweise `nil` zurück, wenn die angegebene URL einen syntaktischen Fehler enthält.

Die Klasse `ClockView` enthält in einem Kommentarblock eine weitere Methode namens `drawRect:`. Entfernen Sie die Kommentarzeichen um diese Methode, da Sie sie im Folgenden überschreiben werden. Cocoa Touch ruft sie automatisch auf, wenn es den Inhalt des Views neu zeichnen muss.

3.1.1 Zeichnen in Cocoa Touch

Als Nächstes erstellen Sie den Code, um alle Bestandteile der Uhr – also Ziffernblatt und die Uhrzeiger – zu zeichnen. Dafür verwenden Sie das *CoreGraphics*-Framework. *CoreGraphics* ist ein sehr mächtiges Framework, das auf reinem C und nicht auf Objective-C basiert. Deshalb enthält es keine Klassen und Methoden, sondern Datentypen, Strukturen und Funktionen. Alle zum *CoreGraphics*-Framework gehörenden Elemente erkennen Sie am Präfix `CG`.

Der Aufbau und die Implementierung der Datentypen in den Frameworks sind vor dem Nutzer versteckt. Sie können auf die Datentypen nur über Referenzen zugreifen. Die Referenzen übergeben Sie an Funktionen, die eine bestimmte Operation ausführen. Auf diese Art implementierte Datentypen nennt man auch *opake Datentypen*.

Der zentrale Datentyp in *CoreGraphics* ist der Grafikkontext, und die Referenzen darauf haben den Typ `CGContextRef`. Ein *Grafikkontext* speichert alle Werte für den aktuellen Zustand Ihrer Zeichenoperationen. Wenn Sie eine Zeichenoperation über einen Grafikkontext ausführen möchten, müssen Sie also zuerst die Parameter wie Farben, Muster und Clipping festlegen und danach die gewünschte Operation aufrufen. Für das Ziffernblatt werden Sie zunächst einen weißen Kreis als Grundfläche zeichnen. Die entsprechenden CoreGraphics-Operationen dazu sehen wie folgt aus:

```
- (void)drawRect:(CGRect)inRectangle {
    CGContextRef theContext = UIGraphicsGetCurrentContext();
    CGRect theBounds = self.bounds;
    CGContextSaveGState(theContext);
    CGContextSetRGBFillColor(theContext, 1.0, 1.0, 1.0, 1.0);
    CGContextAddEllipseInRect(theContext, theBounds);
    CGContextFillPath(theContext);
    CGContextRestoreGState(theContext);
}
```

Listing 3.3 Zeichnen einer weißen Ellipse

Das Rechteck, das die Methode `drawRect:` als Parameter erhält, ist nicht die Zeichenfläche, sondern nur der Bereich der Zeichenfläche, den die Methode neu zeichnen soll. Diesen Wert verwenden Sie allerdings in den meisten Fällen nicht, da dieses Rechteck nur für eine Reduktion der Zeichenoperationen bei optimierter Ausgabe interessant ist. Das ist in der Regel jedoch nur bei sehr großen oder aufwendig zu zeichnenden Views notwendig.

In der ersten Anweisung weist die Methode der Variablen `theContext` über die Funktion `UIGraphicsGetCurrentContext` den aktuellen Grafikkontext zu, in den die Ausgabe erfolgt. Cocoa Touch erzeugt den Kontext vor dem Aufruf der `drawRect:`-Methode, und Sie dürfen `drawRect:` daher niemals direkt aufrufen, da dann kein Kontext zur Verfügung steht. Wenn Sie einen View neu zeichnen lassen möchten, sollten Sie stattdessen die Methode `setNeedsDisplay` des Views aufrufen. Cocoa Touch zeichnet den View dann neu, sobald das möglich ist.

Da die Größe des Ziffernblattes von der Größe des Views abhängt, ermittelt die Methode die Koordinaten und die Größe des Views in der Variablen `theBounds`. Die Koordinaten des Ziffernblattes liegen relativ zu den Koordinaten des Views, deshalb müssen Sie hier die Bounds und nicht den Frame für die Berechnung verwenden.

In der nächsten Zeile sichern Sie den aktuellen Zustand des Grafikkontexts über die Funktion `CGContextSaveGState`. Damit lässt sich am Ende der Methode genau dieser Zustand sehr einfach über die Funktion `CGContextRestoreGState` wiederherstellen. Da Sie nicht wissen, welche Zeichenoperationen in dem Grafikkontext nach der Ausführung Ihrer `drawRect:`-Methode noch ausgeführt werden, sollten Sie immer darauf achten, den Kontext so zu hinterlassen, wie Sie ihn vorgefunden haben. (Ganz anders als die Toiletten der Deutschen Bahn, die man ja so hinterlassen soll, wie man sie vorzufinden *wünscht*. Dummerweise hat man nie einen Dampfstrahler dabei, wenn man ihn braucht.) Achten Sie bei der Verwendung dieser beiden Funktionen aber bitte immer darauf, ihre Aufrufe zu balancieren; das heißt, auf jeden `CGContextSaveGState`-Aufruf sollte immer genau ein `CGContextRestoreGState`-Aufruf folgen.

Die folgenden Zeilen zeichnen dann den Kreis des Ziffernblattes, indem die Methode zuerst die Füllfarbe als *RGBA-Werte* über die Funktion `CGContextSetRGBAFillColor` setzt. Die Fließkommawerte für die Farbkomponenten (Rot, Grün und Blau) und den Alphawert, der die Deckkraft festlegt, dürfen dabei zwischen 0 und 1 liegen. Die Funktion `CGContextAddEllipseInRect` fügt dem Kontext einen Pfad in Form einer Ellipse hinzu. Bei einem quadratischen Ziffernblatt haben die beiden Radien dieser Ellipse die gleiche Länge, und es entsteht ein Spezialfall der Ellipse – ein Kreis. Durch den Aufruf der Funktion `CGContextFillPath` zeichnet schließlich CoreGraphics einen ausgefüllten Kreis mit der gesetzten Füllfarbe.

Pfade in CoreGraphics

Sie zeichnen Linien und Flächen in CoreGraphics, indem Sie im Grafikkontext zuerst den Linienzug der Figur beschreiben. Danach rufen Sie eine Zeichenoperation für den Pfad auf, mit der Sie den Linienzug über `CGContextStrokePath` zeichnen oder ihn über `CGContextFillPath` ausfüllen. Eine weitere mögliche Operation ist das *Clippen* über die Funktion `CGContextClip`. Dabei beschneiden Sie die Zeichenfläche, sodass der Kontext nur noch im Bereich des Linienzuges zeichnet.

Nach der Ausführung der Zeichenoperation vergisst der Kontext übrigens den Pfad. Wenn Sie also einen ausgefüllten Kreis mit einem Rand zeichnen möchten, müssen Sie den Pfad des Kreises jeweils vor den beiden Zeichenoperationen zu dem Kontext hinzufügen.

Wenn Sie Ihr Programm ausführen, werden Sie feststellen, dass immer noch ein weißes Quadrat und ein graues Rechteck angezeigt werden. Das liegt daran, dass der View für das Ziffernblatt ebenfalls Weiß als Hintergrundfarbe verwendet. Er stellt also einen weißen Kreis vor einem weißen Hintergrund dar.

Damit Sie den Kreis sehen können, ändern Sie die Hintergrundfarbe des Views. Öffnen Sie dazu den View im Interface Builder, und wählen Sie den View für das Ziffernblatt aus, indem Sie auf die entsprechende Stelle des Hauptviews klicken. Mit dem Attributinspektor, der zwischen dem Identitäts- und dem Größeninspektor liegt, können Sie über das Attribut BACKGROUND die Hintergrundfarbe des Views verändern (siehe Abbildung 3.11). Wählen Sie ein freundliches Grau oder die Transparenz (CLEAR COLOR) aus. Bei einem transparenten Hintergrund erscheint an den Stellen, die der Kreis nicht verdeckt, die Hintergrundfarbe des umgebenden Views.

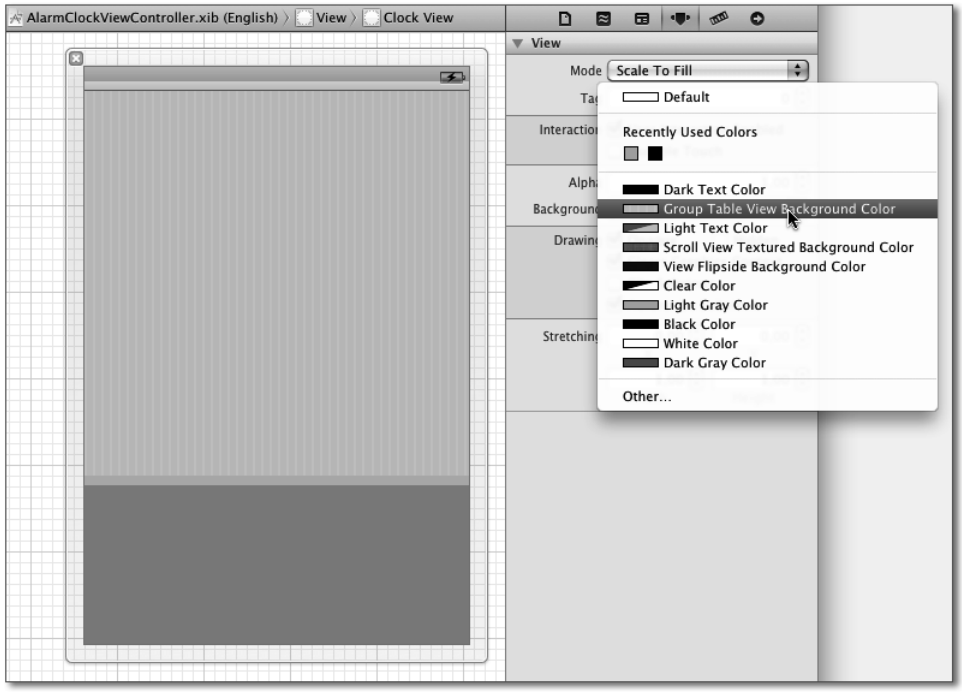

Abbildung 3.11 Der Subview bekommt eine eigene Farbe.

Die Änderung lässt sich umgehend im Simulator überprüfen – das Ziffernblatt ist jetzt sichtbar (siehe Abbildung 3.12) – ein schöner runder Kreis.

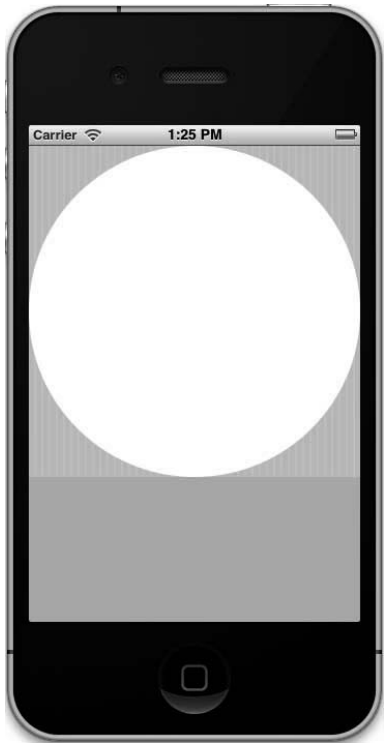

Abbildung 3.12 Das Ziffernblatt ist fast fertig.

Das Ziffernblatt ist in diesem Stadium noch sehr minimalistisch. Im nächsten Schritt fügen Sie die Zeichenoperationen für die Darstellung der Stunden- und Minuteneinteilung hinzu. Dazu braucht die Klasse `ClockView` zwei neue Methoden, die Sie vor der `drawRect:`-Methode einfügen:

```
- (CGPoint)midPoint {
    CGRect theBounds = self.bounds;
    return CGPointMake(CGRectGetMidX(theBounds),
                       CGRectGetMidY(theBounds));
}
```

Listing 3.4 Berechnung des Mittelpunktes des Views

Die Methode `midPoint` berechnet den Mittelpunkt des Views relativ zum `bounds`-Rechteck. Die Klasse `UIView` hat zwar eine ähnliche Methode `center`, die allerdings den Mittelpunkt des `frame`-Rechtecks liefert. Sie ist hierfür jedoch nicht geeignet, da Sie ja das Ziffernblatt relativ zum View zeichnen müssen.

Mathematik für Einsteiger

Sie werden sich möglicherweise fragen, was die viele Mathematik im ersten größeren Beispiel soll. Nun, die Antwort ist einfach. Mathematik begegnet einem Programmierer häufiger, als ihm bisweilen lieb ist. Gerade bei der Erstellung von Frontends und der Arbeit mit Grafiken und Animationen ist in der Regel ein gutes mathematisches Grundverständnis gefragt. Die im Wecker-Beispiel angewandte Mathematik ist ja kein Selbstzweck, sondern resultiert direkt aus der Verwendung des CoreGraphics-Frameworks. Eine kleine Auffrischung Ihres Wissens über Bogenmaß, Winkelberechnung etc. finden Sie bei Wikipedia[3].

Die zweite Methode, `pointWithRadius:angle:`, berechnet einen Punkt, der vom Mittelpunkt eine gegebene Entfernung hat und um einen gegebenen Winkel gedreht ist. Die Winkel werden im Bogenmaß, im Uhrzeigersinn und ausgehend von 12 Uhr gerechnet. Einen Winkel *w* in Grad können Sie über die Formel $w\pi / 180°$ ins Bogenmaß umrechnen.

```
- (CGPoint)pointWithRadius:(CGFloat)inRadius
                    angle:(CGFloat)inAngle {
    CGPoint theCenter = [self midPoint];
    return CGPointMake(theCenter.x + inRadius * sin(inAngle),
                       theCenter.y - inRadius * cos(inAngle));
}
```

Listing 3.5 Berechnung eines Punktes über einen Radius und einen Winkel

Das Ziffernblatt stellt einen halben Tag, also 12 Stunden, dar. Somit entsprechen 12 Stunden einem vollen Kreis von 360°, und eine Stunde entspricht

$360° / 12 = 30°$

oder

$30° \pi / 180° = \pi / 6$

im Bogenmaß. Eine Stunde hat 60 Minuten, und damit hat eine Minute einen Winkel von

$360° / 60 = 6°$

oder

$6° \pi / 180° = \pi / 30$

3 *https://secure.wikimedia.org/wikipedia/de/wiki/Bogenmass*

im Bogenmaß. Für eine Sekunde gelten natürlich die gleichen Winkelwerte wie für Minuten.

Beispielsweise hat ein Zeiger auf »drei Uhr« den Winkel 90° beziehungsweise $\pi / 2$ im Bogenmaß oder auf »10 Minuten« den Winkel 60° beziehungsweise $\pi / 3$. Die Position eines Uhrzeigers oder eines Strichs auf dem Ziffernblatt kann also durch die Angabe eines Winkels und ein oder mehrere Längenangaben bestimmt werden. Diese Beispiele stellt Abbildung 3.13 grafisch dar.

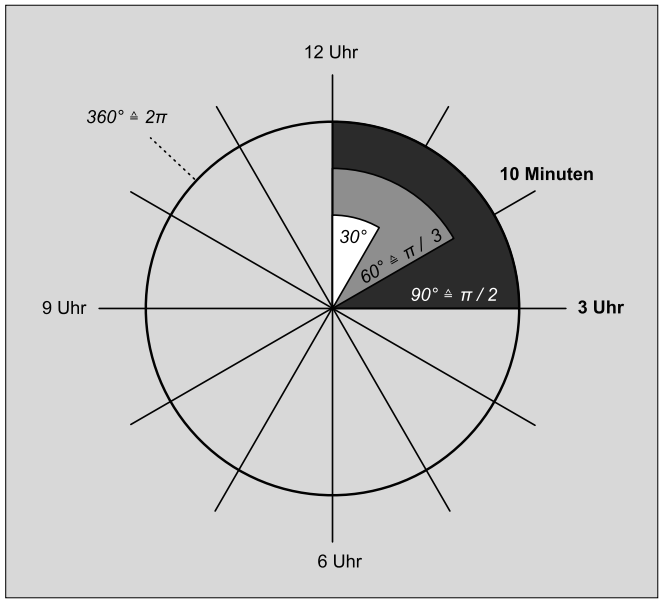

Abbildung 3.13 Beziehung zwischen Uhrzeiten und Winkeln

Das Ziffernblatt soll Minuten durch einen Punkt und Stunden durch einen Strich darstellen. Die Striche für drei, sechs, neun und zwölf Uhr sollen dabei etwas länger als die sonstigen Stundenstriche sein. Die Einteilungen werden über eine Schleife gezeichnet, die von 0 bis 59 läuft; für jede Minute ist jeweils ein Durchlauf vorgesehen.

Da alle Einteilungen die gleiche Farbe, Linienstärke und Linienenden haben sollen, können Sie diese Einstellungen vor der Schleife festlegen. Beachten Sie, dass die Stunden-Einteilungen Linien und die Minuten-Einteilungen Kreise sind. Die Linienfarbe müssen Sie über `CGContextSetRGBStrokeColor` setzen. Die Stundenstriche sollen außerdem abgerundete Linienenden haben, was Sie über die Funktion `CGContextSetLineCap` festlegen können. Da diese runden Kappen jedoch auf beiden Enden der Linien sitzen, reichen sie über den Rand des Ziffernblattes hinaus. Sie können das durch ein *Clipping* verhindern. Das ist eine unsichtbare Maske auf der Zeichenfläche, mit der Sie die Zeichenfläche in Bereiche unterteilen können. Der Grafikkontext zeichnet dann nur in den Bereichen, die zum Clipping gehören. Als Clipping-Region

verwenden Sie den Kreis des Ziffernblattes. Sie fügen dessen Pfad erneut dem Kontext hinzu und setzen diesen in der Methode `drawRect:` durch Aufruf von `CGContextClip` als Clipping-Pfad. Listing 3.6 enthält die soeben beschriebenen Funktionsaufrufe für die Kontexteinstellungen:

```
CGContextSetRGBStrokeColor(theContext, 0.25, 0.25, 0.25, 1.0);
CGContextSetRGBFillColor(theContext, 0.25, 0.25, 0.25, 1.0);
CGContextSetLineWidth(theContext, 7.0);
CGContextSetLineCap(theContext, kCGLineCapRound);
CGContextAddEllipseInRect(theContext, theBounds);
CGContextClip(theContext);
```

Listing 3.6 Setzen von Zeichenattributen im Grafikkontext

Die Schleife zum Zeichnen der Einteilungen sieht nun folgendermaßen aus:

```
CGFloat theRadius = CGRectGetWidth(theBounds) / 2.0;
for(NSInteger i = 0; i < 60; ++i) {
    CGFloat theAngle = i * M_PI / 30.0; // eine Minute
    if(i % 5 == 0) { // alle 5 Minuten ein Strich
        CGFloat theInnerRadius = theRadius *
            (i % 15 == 0 ? 0.7 : 0.8);
        CGPoint theInnerPoint =
            [self pointWithRadius:theInnerRadius
                            angle:theAngle];
        CGPoint theOuterPoint = [self
            pointWithRadius:theRadius
            angle:theAngle];
        CGContextMoveToPoint(theContext, theInnerPoint.x,
                             theInnerPoint.y);
        CGContextAddLineToPoint(theContext, theOuterPoint.x,
                                theOuterPoint.y);
        CGContextStrokePath(theContext);
    }
    else {
        CGPoint thePoint =
            [self pointWithRadius:theRadius * 0.95
                            angle:theAngle];
        CGContextAddArc(theContext, thePoint.x, thePoint.y,
                        3.0, 0.0, 2 * M_PI, YES);
        CGContextFillPath(theContext);
    }
}
```

Listing 3.7 Zeichnen eines Ziffernblattes

Anhand der Schleifenvariablen `i` können Sie entscheiden, um welchen Einteilungstyp es sich handelt. Wenn ihr Wert durch 5 teilbar ist (`i % 5 == 0`), muss es ein Strich sein. Ist sie außerdem auch durch 15 teilbar (`i % 15 == 0`), so muss es ein langer Strich sein. Alle anderen Werte sind Punkte. Die Länge der Striche und die Lage der Punkte errechnen sich relativ zum Radius des Ziffernblattes.

Um die Linien jeweils am gewünschten Punkt beginnen zu lassen, müssen Sie den Startpunkt der Linie setzen, ohne den Pfad des Kontexts zu erweitern. Dazu verwenden Sie die Funktion `CGContextMoveToPoint`.

Teilbarkeit

Ob eine ganze Zahl ein Teiler einer anderen Zahl ist, können Sie über den *Divisionsrestoperator* `%` ermitteln. Wenn der Rest einer Division 0 ist, dann ist der Dividend durch den Divisor teilbar. Beispielsweise prüft der Ausdruck `i % 5 == 0`, ob der Dividend `i` durch den Divisor 5 teilbar ist.

Zum Zeichnen der Punkte verwendet der Code die Funktion `CGContextAddArc`. Es ist hier einfacher, den Kreis über dessen Mittelpunkt und Radius zu bestimmen als über sein umgebendes Rechteck. Über diese Funktion können Sie beliebige Kreisbögen erzeugen, weswegen Sie den Start- und den Endwinkel des Bogens im Bogenmaß angeben müssen. Damit Sie prüfen können, ob Sie auf dem richtigen Weg sind, sehen Sie hier noch einmal die ganze Methode `drawRect:` an einem Stück, und die Ausgabe sehen Sie in Abbildung 3.14.

```
- (void)drawRect:(CGRect)inRectangle {
    CGContextRef theContext = UIGraphicsGetCurrentContext();
    CGRect theBounds = self.bounds;
    CGFloat theRadius = CGRectGetWidth(theBounds) / 2.0;

    CGContextSaveGState(theContext);
    CGContextSetRGBFillColor(theContext, 1.0, 1.0, 1.0, 1.0);
    CGContextAddEllipseInRect(theContext, theBounds);
    CGContextFillPath(theContext);
    CGContextAddEllipseInRect(theContext, theBounds);
    CGContextClip(theContext);
    CGContextSetRGBStrokeColor(theContext, ➲
        0.25, 0.25, 0.25, 1.0);
    CGContextSetRGBFillColor(theContext, ➲
        0.25, 0.25, 0.25, 1.0);
    CGContextSetLineWidth(theContext, 7.0);
    CGContextSetLineCap(theContext, kCGLineCapRound);
    for(NSInteger i = 0; i < 60; ++i) {
```

```
        CGFloat theAngle = i * M_PI / 30.0;

        if(i % 5 == 0) {
            CGFloat theInnerRadius = theRadius * ➲
                (i % 15 == 0 ? 0.7 : 0.8);
            CGPoint theInnerPoint = ➲
                [self pointWithRadius:theInnerRadius ➲
                                angle:theAngle];
            CGPoint theOuterPoint = ➲
                [self pointWithRadius:theRadius ➲
                                angle:theAngle];
            CGContextMoveToPoint(theContext, theInnerPoint.x, ➲
                                 theInnerPoint.y);
            CGContextAddLineToPoint(theContext, ➲
                                    theOuterPoint.x,
                                    theOuterPoint.y);
            CGContextStrokePath(theContext);
        }
        else {
            CGPoint thePoint =
                [self pointWithRadius:theRadius * 0.95 ➲
                                angle:theAngle];
            CGContextAddArc(theContext,thePoint.x, thePoint.y,➲
                            3.0, 0.0, 2 * M_PI, YES);
            CGContextFillPath(theContext);
        }
    }
    [self drawClockHands];
    CGContextRestoreGState(theContext);
}
```

Listing 3.8 Die komplette Methode zum Zeichnen des Ziffernblattes

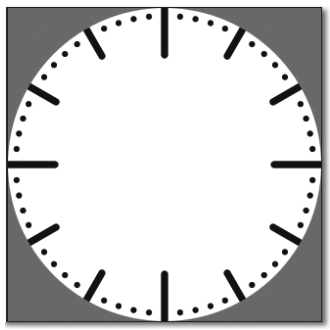

Abbildung 3.14 Das Ziffernblatt des Weckers

Die Methode `drawClockHands`, deren Aufruf am Ende von `drawRect:` erfolgt, zeichnet die Zeiger des Weckers. Für die Implementierung müssen Sie die aktuelle Systemzeit in Winkel umrechnen, damit Sie die Zeiger durch Linien darstellen können.

3.1.2 Zeitberechnung

Der `ClockView` soll die angezeigte Zeit in der Property `time` speichern. Diese hat den Typ `NSDate`, dessen Objekte einen Zeitpunkt enthalten, der auf einem Referenzzeitpunkt basiert. Da die Property das Datumsobjekt halten muss, bekommt sie den Speicherverwaltungstyp `retain`. Das Datumsobjekt enthält den Zeitpunkt als Anzahl der Sekunden zu einem festgelegten Referenzzeitpunkt – dem 1. Januar 2001 in der Zeitzone *Greenwich Mean Time (GMT)*. Hingegen ist die Umrechnung des Zeitpunktes in eine Datumsdarstellung sehr schwer. Dafür gibt jedoch es glücklicherweise spezielle Klassen. Diese Darstellung von Datumswerten klingt zunächst unnötig kompliziert, hat indes einige Vorteile:

- Sie können mit Datumsobjekten gut rechnen. Beispielsweise können Sie einen weiteren Zeitpunkt durch Addition eines positiven oder negativen Zeitintervalls berechnen.
- Der Wert eines Zeitpunktes hat überall auf der Welt und zu jeder Zeit die gleiche Bedeutung.
- Es gibt sehr viele unterschiedliche Zeitzonen und Kalender, wie beispielsweise den buddhistischen, gregorianischen und japanischen Kalender. Es gibt also sehr viele verschiedene Darstellungen des gleichen Zeitpunktes. Es ist einfacher, einen Zeitpunkt in eine Datumsdarstellung umzuwandeln als eine Datumsdarstellung in eine andere.
- Die internen Uhren aller Computer – auch die der iPhones und iPads – und der Atomuhren stellen die aktuelle Zeit als Differenz zu einem Referenzzeitpunkt dar. Für die Erzeugung eines Datumsobjekts braucht der Computer also nicht viel Rechenzeit.

Um einen Zeitpunkt in eine Datumsdarstellung umzurechnen, gibt es zwei Klassen im Foundation-Framework:

- `NSDateFormatter`
- `NSCalendar`

Mit `NSDateFormatter` können Sie zu einem `NSDate`-Objekt eine Zeichenkette erzeugen, die das enthaltene Datum darstellt. Diese Klasse haben Sie bereits im zweiten Kapitel verwendet, und Sie kommt auch im Beispielprojekt *Fototagebuch* zum Einsatz.

Mit Objekten der Klasse `NSCalendar` berechnen Sie basierend auf einer Zeitzone und einer Region die Datumskomponenten zu einem Zeitpunkt und umgekehrt. Mit die-

ser Klasse und ein bisschen Rechnerei können Sie also die Positionen der Uhrzeiger zu einem Zeitpunkt bestimmen. Aus diesem Grund enthält die Klasse eine weitere, haltende Property namens `calendar` vom Typ `NSCalendar`. Mit der Methode `components:fromDate:` aus der Kalenderklasse können Sie ein `NSDateComponents`-Objekt berechnen lassen, das die benötigten Zeitkomponenten (Stunde, Minute und Sekunde) enthält. Mit den oben angegebenen Formeln können Sie daraus dann die entsprechenden Winkel für die Darstellung bestimmen.

Wenn Sie allerdings die Stunde in einen Winkel umrechnen, bekommen Sie einen Stundenzeiger, der immer nur auf volle Stunden zeigt. Bei einer analogen Uhr soll er jedoch beispielsweise um halb fünf genau zwischen der vierten und fünften Stunde stehen. Das können Sie erreichen, indem Sie zum Stundenwert den Minutenanteil hinzurechnen. Da eine Stunde bekanntlich 60 Minuten hat, ist der Minutenanteil einer Stunde ein Sechzigstel des Minutenwertes.

```
NSDateComponents *theComponents =
    [self.calendar components:NSHourCalendarUnit |➲
                              NSMinuteCalendarUnit |➲
                              NSSecondCalendarUnit➲
                      fromDate:self.time];
CGFloat theSecond = theComponents.second * M_PI / 30.0;
CGFloat theMinute = theComponents.minute * M_PI / 30.0;
CGFloat theHour = (theComponents.hour + ➲
                   theComponents.minute / 60.0) * M_PI / 6.0;
```

Listing 3.9 Berechnung der Winkel für Sekunde, Minute und Stunde

Die Variablen `theSecond`, `theMinute` und `theHour` enthalten den Winkel für die entsprechende Zeigerstellung. Der Faktor $\pi / 30 = 2\pi / 60$ in der Berechnung entspricht dabei dem Winkel im Bogenmaß von einer Sekunde beziehungsweise einer Minute. Analog ist $\pi / 6 = 2\pi / 12$ der Winkel, der einer vollen Stunde auf einem Ziffernblatt entspricht.

Ist er Ihr Typ?

Bei der Berechnung des Minutenanteils verwendet Listing 3.9 nicht `60`, sondern `60.0` als Teiler. Der Unterschied dieser beiden Werte ist ihr Datentyp. Während `60` den Typ `int` hat, ist `60.0` ein `double`. Da der Minutenwert `theComponents.minute` den Typ `NSInteger` hat, würde bei einer Division durch `60` eine Ganzzahldivision ausgeführt. Der Minutenwert liegt immer zwischen 0 und 59, bei einer Ganzzahldivision kommt dabei also immer – *Tusch* – 0 heraus. Die Rechnung können Sie sich also auch sparen. Bei einer Fließkommadivision erhalten Sie hingegen Werte zwischen 0 und 0,99.

Jetzt brauchen Sie nur noch die drei Zeiger zu zeichnen, indem Sie eine Linie vom Mittelpunkt des Views in Richtung des jeweiligen Winkels ziehen. Die drei Linien unterscheiden sich jedoch nicht nur hinsichtlich ihres Winkels, sondern auch in der Länge, der Farbe und der Linienstärke.

Den Zielpunkt der Zeiger berechnen Sie jeweils über die Methode `pointWithRadius:angle:`, wobei Sie die unterschiedlichen Längen durch die Verwendung unterschiedlicher Radien erzielen. Das Zeichnen der einzelnen Zeiger ist recht ähnlich und unterscheidet sich nur in den unterschiedlichen Zeichenattributen für die Linien. Die gesamte Methode `drawClockHands` sieht dann wie folgt aus:

```
- (void)drawClockHands {
    CGContextRef theContext = UIGraphicsGetCurrentContext();
    CGPoint theCenter = [self midPoint];
    CGFloat theRadius = CGRectGetWidth(self.bounds) / 2.0;
    NSDateComponents *theComponents = ➲
        [self.calendar components:NSHourCalendarUnit  ➲
                                  NSMinuteCalendarUnit  ➲
                                  NSSecondCalendarUnit  ➲
                          fromDate:self.time];
    CGFloat theSecond = theComponents.second * M_PI / 30.0;
    CGFloat theMinute = theComponents.minute * M_PI / 30.0;
    CGFloat theHour = (theComponents.hour +  ➲
        theComponents.minute / 60.0) * M_PI / 6.0;
    CGPoint thePoint = [self pointWithRadius:theRadius * 0.7  ➲
                                       angle:theHour];

    CGContextSetRGBStrokeColor(theContext,➲
                               0.25, 0.25, 0.25, 1.0);
    CGContextSetLineWidth(theContext, 7.0);
    CGContextSetLineCap(theContext, kCGLineCapButt);
    CGContextMoveToPoint(theContext, theCenter.x, theCenter.y);
    CGContextAddLineToPoint(theContext, thePoint.x, thePoint.y);
    CGContextStrokePath(theContext);
    thePoint = [self pointWithRadius:theRadius * 0.9  ➲
                               angle:theMinute];
    CGContextSetLineWidth(theContext, 5.0);
    CGContextMoveToPoint(theContext, theCenter.x, theCenter.y);
    CGContextAddLineToPoint(theContext, thePoint.x, thePoint.y);
    CGContextStrokePath(theContext);
    thePoint = [self pointWithRadius:theRadius * 0.95  ➲
                               angle:theSecond];
    CGContextSetLineWidth(theContext, 3.0);
    CGContextSetRGBStrokeColor(theContext, 1.0, 0.0, 0.0, 1.0);
```

```
    CGContextMoveToPoint(theContext, theCenter.x, theCenter.y);
    CGContextAddLineToPoint(theContext, thePoint.x, thePoint.y);
    CGContextStrokePath(theContext);
}
```

Listing 3.10 So werden die Uhrzeiger des Weckers gezeichnet.

3.1.3 View-Erzeugung über NIB-Dateien

Sie müssen den beiden Propertys `calendar` und `time` vor der ersten Anzeige des Views nun noch sinnvolle Werte zuweisen. Dazu können Sie die Initialisierungsmethoden entsprechend erweitern. Die betreffenden Werte müssen Sie in der Methode `dealloc` natürlich auch wieder freigeben, indem Sie die Propertys auf `nil` setzen.

Der Convenience-Konstruktor `date` der Klasse `NSDate` erzeugt ein Datumsobjekt mit der aktuellen Uhrzeit in GMT, und `currentCalendar` liefert ein Kalenderobjekt, das den aktuellen Einstellungen des iPhones entspricht.

```
- (id)initWithFrame:(CGRect)inFrame {
    self = [super initWithFrame:inFrame];
    if(self) {
        self.calendar = [NSCalendar currentCalendar];
        self.time = [NSDate date];
    }
    return self;
}
- (void)dealloc {
    self.calendar = nil;
    self.time = nil;
    [super dealloc];
}
```

Listing 3.11 Initialisierung und Freigabe der Propertys der Zeitanzeige

Ihr Programm verwendet die Initialisierungsmethode `initWithFrame:` allerdings nur, wenn Sie Ihren View direkt über Programmcode erzeugen. Die App lädt diesen View jedoch aus einer NIB-Datei oder einem Storyboard und ruft dabei diese Methode nicht auf.

Für diesen Fall gibt es zwei mögliche Alternativen. Sie können eine weitere Initialisierungsmethode namens `initWithCoder:` oder eine Methode namens `awakeFromNib` implementieren. Sie dürfen dabei eine oder auch beide Methoden verwenden. Die Initialisierung eines Objekts aus einer NIB-Datei erfolgt über `initWithCoder:`, sofern sie diese Methode besitzt. Bei allen anderen Klassen kommt die Initialisierungsmethode `init` zum Einsatz. Da die Klasse `UIView` das Protokoll `NSCoder` adaptiert und

somit diesen Initializer bereitstellt, erfolgt die Initialisierung immer darüber. Sind alle Objekte aus der NIB-Datei geladen und initialisiert, sendet Cocoa Touch an jedes Objekt die Nachricht `awakeFromNib`, die es von `NSObject` erbt. Diese Methode können Sie also überschreiben, um ein Objekt erst dann zu initialisieren, wenn Cocoa Touch alle Objekte aus der NIB-Datei geladen hat. Den Ablauf für die Initialisierung eines Views stellt Abbildung 3.15 grafisch dar.

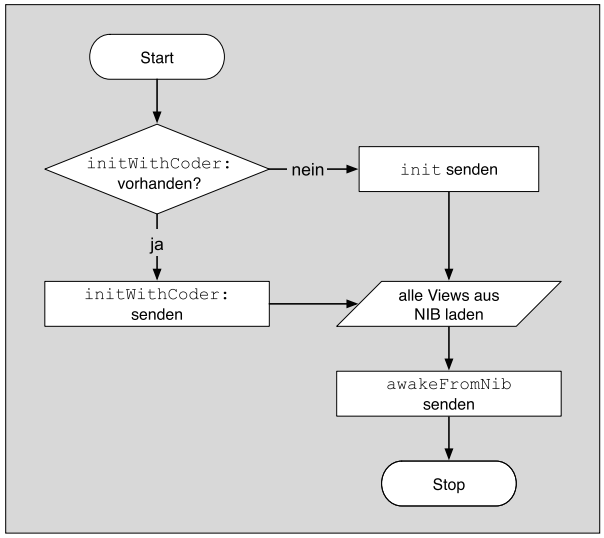

Abbildung 3.15 Ablauf der Initialisierung für Views aus NIB-Dateien

Wie bereits weiter oben erwähnt wurde, sind alle Objekte in einer NIB-Datei als serialisierte Daten abgelegt. Die *Serialisierung* enthält alle notwendigen Informationen, um diese Objekte wiederherzustellen. Zu diesen Informationen gehören die Klassen, die Attributwerte und die Anordnung der Objekte zueinander. Während NIB-Dateien dafür ein kompaktes, binäres Dateiformat verwenden, verwenden XIB-Dateien und Storyboards ein XML-Format.

Wenn die Deserialisierung die Methode `initWithCoder:` des neuen Objekts aufruft, übergibt sie einen Parameter vom Typ `NSCoder` mit den notwendigen Daten für dieses Objekt. Da Ihre Klasse jedoch die Serialisierung nicht erweitert, brauchen Sie den Parameter nur für den Initialisierer-Aufruf der Oberklasse. Wenn Sie die Methode `initWithCoder:` implementieren, sollten Sie immer als Erstes diese Methode der Oberklasse so wie in Listing 3.12 aufrufen:

```
- (id)initWithCoder:(NSCoder *)inCoder {
    self = [super initWithCoder:inCoder];
    if(self) {
        self.calendar = [NSCalendar currentCalendar];
        self.time = [NSDate date];
```

```
    }
    return self;
}
```

Listing 3.12 Initialisierung der Propertys bei der Deserialisierung

Nach der Erzeugung aller Objekte aus einer NIB-Datei ruft Cocoa Touch die Methode `awakeFromNib` bei allen erzeugten Objekten auf. Wenn Sie diese Methode überschreiben, sollten Sie auch immer als Erstes die Methode in der Oberklasse aufrufen.

```
- (void)awakeFromNib {
    [super awakeFromNib];
    self.calendar = [NSCalendar currentCalendar];
    self.time = [NSDate date];
}
```

Listing 3.13 Initialisierung der Propertys nach der Deserialisierung

Verlassen Sie sich weder bei `initWithCoder:` noch bei `awakeFromNib` auf eine feste Aufrufreihenfolge. Für die Initialisierung ist häufig `awakeFromNib` die geeignetere Wahl, da bei Aufruf dieser Methode auch alle anderen Objekte des NIB-Files die Initialisierung durchlaufen haben.

3.1.4 Aktualisierung der Zeitanzeige

Als Nächstes soll die Zeitanzeige kontinuierlich aktualisiert werden. Diese Aufgabe kann entweder der View selbst oder der Viewcontroller übernehmen. Da jedoch jedes Objekt der Klasse `ClockView` diese Aufgabe erfüllen sollte, wird der dafür nötige Code in der View-Klasse angesiedelt.

Für die Aktualisierung der Zeitanzeige müssen Sie nur in regelmäßigen Abständen den Wert der Property `time` aktualisieren. Wenn Sie eine kurze Aufgabe wiederholt ausführen möchten, können Sie dafür die Klasse `NSTimer` verwenden.

Die Klasse `ClockView` erhält dazu eine weitere Property von diesem Typ und jeweils eine Methode zum Starten und zum Stoppen der Aktualisierung.

Anonyme Kategorie

Die Klasse verwendet die Property `timer` nur intern. Eine Zugriffsmöglichkeit von außen ist also nicht notwendig. Das lässt sich durch eine *anonyme Kategorie* vermeiden. Bei einer anonymen Kategorie geben Sie bei der Deklaration zwischen den runden Klammern keinen Namen an, und als Implementierungsteil verwenden Sie den der Klasse.

In der Regel deklarieren Sie die anonyme Kategorie einer Klasse in deren Implementierungsdatei. Sie können die Timer-Property also in der anonymen Kategorie deklarieren, da sie für andere Klassen nicht sichtbar sein soll:

```
@interface ClockView()
@property(nonatomic, retain) NSTimer *timer;
@end
@implementation ClockView
@synthesize timer;
...
@end
```

Die neueren Xcode Versionen legen übrigens bei Viewcontroller-Klassen automatisch jeweils eine anonyme Kategorie an.

Sie können so die Property vor dem allgemeinen Zugriff verstecken. In der anonymen Kategorie können Sie sämtliche Methoden und Propertys deklarieren, die die Klasse `ClockView` nur intern braucht und die sie nicht für die Öffentlichkeit zur Verfügung stellt. Die Verwaltung des Property-Inhalts erfolgt über die beiden Methoden `startAnimation` und `stopAnimation`, auf die auch andere Klassen zugreifen dürfen.

```
@interface ClockView : UIView
...
- (void)startAnimation;
- (void)stopAnimation;
@end
```

Listing 3.14 Deklarationen für die automatische Aktualisierung der Zeitanzeige

Ein *Timer* kann eine Methode in einem Objekt nach einer festen Zeitspanne aufrufen. Er kann das einmal oder auch kontinuierlich machen. In der Methode `startAnimation` müssen Sie also ein Timer-Objekt erzeugen und starten, wogegen Sie es in `stopAnimation` anhalten und zerstören. Sie können über die Property für den Timer außerdem verhindern, dass die App den Timer versehentlich mehrmals startet. Dazu prüft die Methode in Listing 3.15 über die Bedingung `self.timer == nil`, ob die Property `timer` nicht schon etwa auf einen Timer verweist:

```
- (void)startAnimation {
    if(self.timer == nil) {
        self.timer = [NSTimer➲
            scheduledTimerWithTimeInterval:0.5➲
            target:self selector:@selector(updateTime:)➲
            userInfo:nil repeats:YES];
    }
}
```

```
- (void)stopAnimation {
    [self.timer invalidate];
    self.timer = nil;
}
```

Listing 3.15 Starten und Stoppen der Animation

Der Aufruf von `scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:` in der dritten Zeile erzeugt und startet einen kontinuierlichen Timer und weist ihn der Property `timer` zu. Dabei geben der zweite und der dritte Parameter an, dass der Timer die Methode `updateTime:` im aktuellen Objekt aufrufen soll. Der Doppelpunkt im Selektor zeigt an, dass diese Methode einen Parameter erwartet. Das Zeitintervall beträgt eine halbe Sekunde, damit der Timer die Anzeige zweimal pro Sekunde aktualisiert. Da die Methodenaufrufe nicht in exakten Zeitabständen erfolgen, könnten bei längeren Zeitintervallen (z. B. einer Sekunde) sonst Sprünge auftreten, was zu ruckelnden Zeigerbewegungen führt.

Anonyme Kategorien erlauben auch das Überschreiben der Read-Only-Eigenschaft von Propertys. Dazu müssen Sie die Property, die Sie als `readonly` in der Schnittstelle der Klasse deklariert haben, genau mit den gleichen Parametern und dem Parameter `readwrite` in der anonymen Kategorie deklarieren. Sie dürfen keinen Parameter – bis auf `readonly` in `readwrite` – bei der Redeklaration verändern. Außerdem müssen Sie natürlich im Header auch den gleichen Speicherverwaltungstyp angeben. Die Property `calendar` der Klasse `ClockView` soll von außen unveränderbar sein:

```
// Deklaration in ClockView.h
@property (nonatomic, retain, readonly) ↵
    NSCalendar *calendar;
// Deklaration in der anonymen Kategorie in ClockView.m
@property (nonatomic, retain, readwrite) ↵
    NSCalendar *calendar;
```

Listing 3.16 Öffentliche Read-Only- und private Read-Write-Property

Sie können nach dieser Änderung in der Implementierung von `ClockView` die Property `calendar` wie bisher verwenden. Außerhalb der Implementierung können Sie die Property dagegen jetzt nur noch auslesen. Eine Veränderung des Wertes ist nicht mehr möglich.

Tipp

Verwenden Sie anonyme Kategorien, um Implementierungsdetails Ihrer Klassen nach außen hin zu verstecken. Anonyme Kategorien erlauben Ihnen eine klarere Strukturierung Ihrer Schnittstellen.

Die Methode `updateTime:` aktualisiert den Wert der Property `time`. Damit Cocoa Touch den View danach auch neu zeichnet, müssen Sie auch noch die Methode `setNeedsDisplay` aufrufen. Sie zeichnet den View allerdings nicht direkt neu, sondern markiert ihn nur zum Neuzeichnen. Das macht das Betriebssystem erst dann, wenn es dafür Zeit hat.

```
- (void)updateTime:(NSTimer *)inTimer {
    self.time = [NSDate date];
    [self setNeedsDisplay];
}
```

Listing 3.17 Aktualisierung der Zeitanzeige

Automatische Aktualisierung des Views versus Animationen

Die in diesem Abschnitt vorgestellten Methoden dienen zur automatischen Aktualisierung eines Views. Dieses Vorgehen ist für diesen Anwendungsfall auch durchaus ausreichend, da Sie die Zeiger nur zweimal pro Sekunde neu zeichnen müssen. Bei jeder Aktualisierung zeichnet der View nicht nur die Zeiger, sondern auch das komplette Ziffernblatt neu. Die Uhr läuft trotzdem flüssig.

Dieses Vorgehen eignet sich allerdings nicht für beliebige Animationen, da dort in der Regel häufigere Aktualisierungen des Views notwendig sind. Außerdem gibt Ihnen Cocoa Touch dafür mit *Core Animation* ein viel besseres System an die Hand, und einfache Animationen lassen sich außerdem problemlos über Klassenmethoden von `UIView` realisieren. Mehr dazu finden Sie in Kapitel 6, »Models, Layer, Animationen«.

3.1.5 Wiederverwendbarkeit von Views

Die Klasse `ClockView` weist bis jetzt keine Abhängigkeiten vom Viewcontroller oder vom Application-Delegate auf. Sie ist also vollkommen unabhängig von der Controller- oder sogar der Modellschicht des Programms. Das ist so gewollt und soll auch so bleiben. Die Unabhängigkeit erlaubt es Ihnen, beliebig viele Uhren in Ihrem Projekt anzulegen.

Sie können also erneut die XIB-Datei öffnen und weitere ClockView-Objekte so, wie oben beschrieben, anlegen. Sie brauchen dazu einfach nur im Interface Builder weitere `UIView`-Objekte auf den Hauptview ziehen und die Klasse der neuen Objekte auf `ClockView` zu setzen. Sie können sogar die Header- und die Implementierungsdatei der Klasse in ein anderes Projekt übertragen und den View dort ohne Einschränkungen oder weitere notwendige Schritte verwenden. Die Größen der Uhren lassen sich über den Größeninspektor auch beliebig anpassen. Sie können sogar

Ziffernblätter erzeugen, die nicht kreis-, sondern ellipsenförmig sind. Letztere sehen allerdings etwas gewöhnungsbedürftig aus. Außerdem können Sie die Farbe für den Hintergrund hinter dem Ziffernblatt frei wählen. Ein Beispiel dafür sehen Sie in Abbildung 3.16.

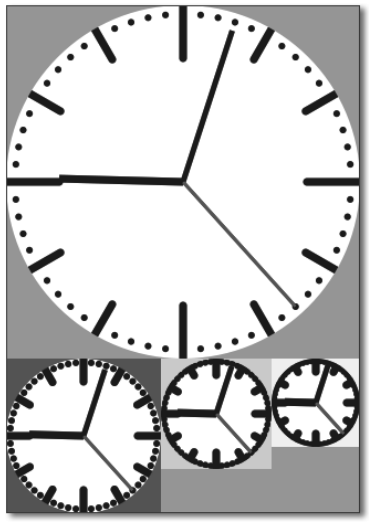

Abbildung 3.16 Wiederverwendung eines Views

Ein möglichst hoher Wiederverwendungsgrad sollte immer Ihr Ziel bei der Entwicklung eigener View-Komponenten sein. Zugegebenermaßen kann der View auch nur eine Uhrzeit darstellen. Sie lernen im Verlauf dieses Buches noch weitere Möglichkeiten kennen, wie der View mit den Objekten aus den anderen Schichten interagieren kann, ohne dass dabei Abhängigkeiten zu Klassen aus diesen Schichten entstehen.

3.2 Views und Viewcontroller

Das Beispielprojekt *Clock* zeigt zwar mehrere Uhren an, deren Zeiger bleiben aber stehen. Das liegt daran, dass dieses Programm nicht die Methode `startAnimation` aufruft. Wenn Sie diesen Aufruf in die Methode `awakeFromNib` in der Klasse `ClockView` einfügen, bewegen sich die Zeiger. Allerdings startet dann das Laden des Views die Timer, und erst der Aufruf der Methode `dealloc` der Uhren stoppt sie. Die Uhren laufen also ständig. Bei so einer kleinen App, die nur eine Ansicht zeigt, ist das zwar nicht schlimm. In größeren Projekten ist es sicherlich sinnvoller, die Timer nur dann laufen zu lassen, wenn die App sie auch tatsächlich anzeigt. Die Klasse `UIView` bietet dafür jedoch keine entsprechenden Methoden an, und dafür ist sie auch nicht da.

Für die generellen Abläufe in der Applikation ist schließlich die Controllerschicht zuständig. Und deshalb sollte der Viewcontroller diese Aufgabe übernehmen. In

Cocoa Touch leiten sich die Viewcontroller von der Klasse `UIViewController` ab. Sie bietet vier Methoden an, die sie bei einer Änderung des Anzeigezustandes aufruft und die Sie in Ihren Unterklassen überschreiben können. Die Methoden heißen `viewWillAppear:`, `viewDidAppear:`, `viewWillDisappear:` und `viewDidDisappear:` und haben jeweils einen booleschen Parameter.

Während Cocoa Touch die ersten beiden Methoden bei der Anzeige des Views aufruft, ruft es die letzten beiden beim Verschwinden des Views auf. Der Namensbestandteil *Will* bedeutet, dass Cocoa Touch die Methode vor dem Ereignis aufruft, während *Did* darauf hinweist, dass Cocoa Touch die Methode nach dem Ereignis aufruft. Der boolesche Parameter gibt dabei jeweils an, ob die Anzeige oder das Verschwinden animiert erfolgt. Wenn Sie diese Methoden überschreiben, müssen Sie auch immer die entsprechende Methode in der Oberklasse aufrufen.

Sie könnten jetzt also die Uhren in `viewDidAppear:` starten und in `viewWillDisappear:` stoppen, und stellen damit sicher, dass die Uhr immer läuft, wenn sie die Applikation anzeigt. Sie müssen dazu nur noch Ihren Viewcontroller mit den Uhren bekannt machen.

Sie können in NIB-Dateien nicht nur den View-Aufbau speichern, sondern auch Verbindungen zwischen den Objekten im NIB und dem *Eigentümer* der NIB-Datei herstellen. (Der Eigentümer wird auch *File's Owner* genannt.)

Der Eigentümer ist dabei das Objekt, das die Datei geöffnet hat. Das ist in der Regel der Viewcontroller. Es gibt vier Verbindungstypen:

- *Actions* sind Verbindungen zwischen Views, die Ereignisse versenden, und Methoden, die diese Ereignisse verarbeiten. Damit können Sie beispielsweise einen Methodenaufruf auslösen, wenn Sie einen Button drücken. Dieser Verbindungstyp ist Gegenstand des Abschnitts 3.2.5.
- Ein *Outlet* verbindet ein Objekt in der NIB-Datei mit einem Attribut oder einer Property eines anderen Objekts. Auf der Seite des Eigentümers ist ein *Outlet* einfach ein Attribut oder besser eine Property, mit denen Sie dann auf das Objekt zugreifen können. Im zweiten Kapitel haben Sie zum Beispiel einen Textview über ein Outlet mit dem Viewcontroller verbunden.
- Eine *Outlet-Collection* verbindet mehrere Objekte mit einem Array-Attribut[4] oder einer Array-Property. Sie können über Outlet-Collections also auf mehrere gleichartige Objekte des NIBs zugreifen. Diesen Verbindungstyp gibt es erst seit iOS 4.0.
- Ein *Segue* (*Übergang*) verbindet in einem Storyboard unter iOS 5 jeweils einen Viewcontroller oder einen View, der Ereignisse versendet, mit einem Viewcontroller. Der Übergang beschreibt dabei den Wechsel von einem Viewcontroller zu einem anderen. Übergänge werden in Abschnitt 3.2.9 besprochen.

4 Cocoa Touch erlaubt zwar Outlets und Outlet-Collections über Ivars. Da Sie jedoch eine etwas eigentümliche Speicherverwaltung haben, sollten Sie lieber Propertys bevorzugen.

3.2.1 Outlets

Outlets erzeugen Sie über den Interface Builder. Öffnen Sie dazu den View des Beispielprojekts, indem Sie die Datei *AlarmClockViewController.xib* auswählen. Aktivieren Sie in der Werkzeugleiste den Hilfseditor durch den mittleren Button in der Gruppe EDITOR (in Abbildung 3.17 oben rechts). Der Hilfseditor sollte nun die Headerdatei *AlarmClockViewController.h* des Viewcontrollers anzeigen. Sie können die angezeigte Datei über die Menüleiste des Editors unterhalb der Werkzeugleiste ändern, falls der Editor eine andere Datei geöffnet haben sollte.

Wenn Sie jetzt mit der rechten Maustaste (oder mit der linken bei gedrückter ctrl-Taste) auf das Ziffernblatt klicken, können Sie eine Verbindung von dem View zu der Headerdatei des Controllers ziehen, wie Abbildung 3.17 zeigt.

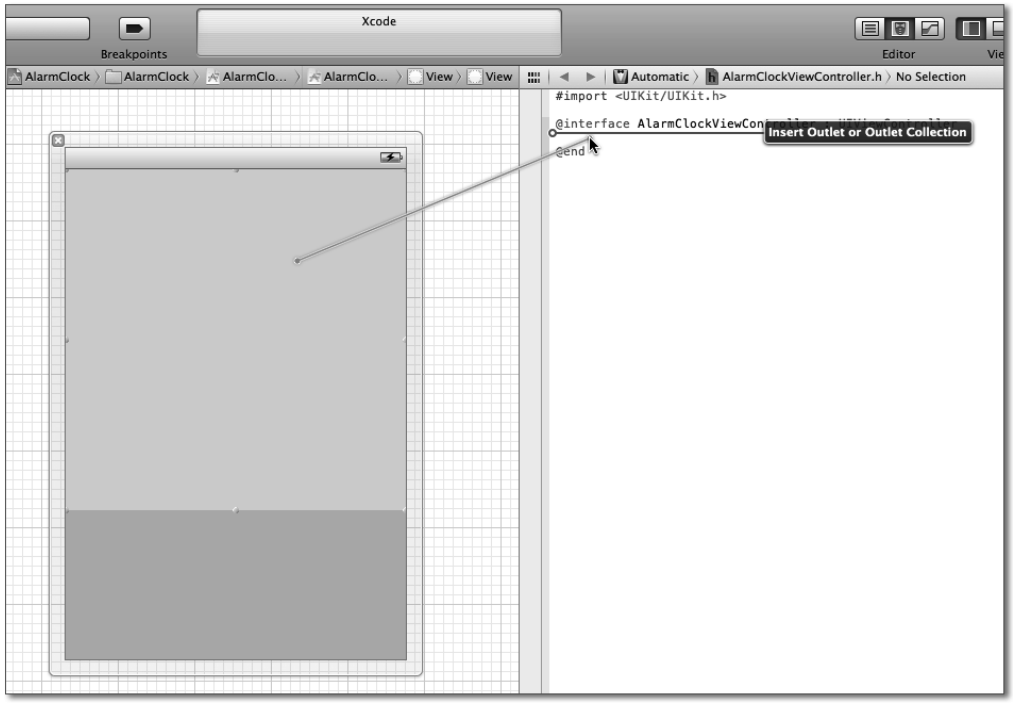

Abbildung 3.17 Ziehen einer Outlet-Verbindung

Nachdem Sie die Maustaste innerhalb der Klassendeklaration losgelassen haben, erscheint der in Abbildung 3.18 abgebildete Pop-over-Dialog, über den Sie ein Outlet oder eine Outlet-Collection anlegen können. Geben Sie `clockView` in das Textfeld ein, und lassen Sie die übrigen Einstellungen des Dialogs unverändert.

Durch einen Klick auf den Button CONNECT erstellen Sie im Quelltext ein Attribut und eine Property, vor deren Klasse die Markierung `IBOutlet` steht. Diese Markierung ist ein leeres C-Makro, das keine Auswirkung auf den erzeugten Programmcode hat.

Es dient lediglich dazu, dem Interface Builder anzuzeigen, dass er zu dieser Property Verbindungen herstellen kann.

```
@property (nonatomic, retain) IBOutlet ClockView *clockView;
```

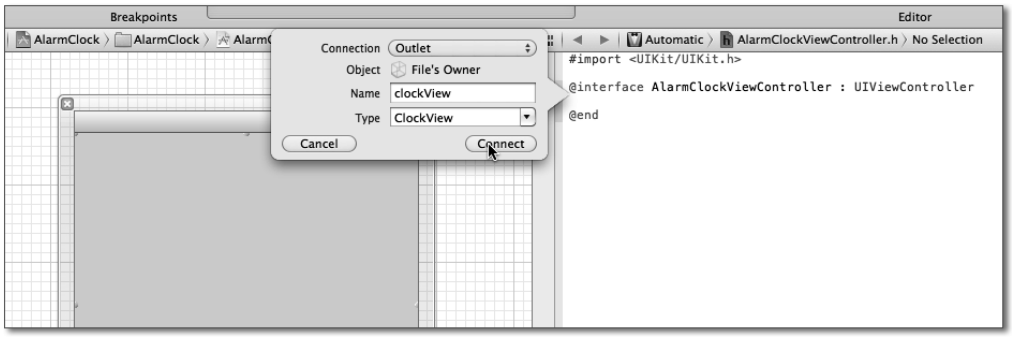

Abbildung 3.18 Pop-over-Dialog für das Outlet

Der Speicherverwaltungstyp der Property hängt davon ab, welches Speicherverwaltungsmodell Ihr Projekt verwendet. Bei manuellem Referenzenzählen hat die Property den Typ `retain`, und bei automatischem Referenzenzählen hat sie den Typ `strong` oder optional `weak`. Außerdem müssen Sie die Klasse `ClockView` noch im Header bekannt machen. Dazu können Sie entweder deren Header-Datei über die Anweisung `#import "ClockView.h"` importieren oder eine Vorwärtsdeklaration mit der Anweisung `@class ClockView;` vor der Deklaration der Controllerklasse einfügen. Eine *Vorwärtsdeklaration* sagt dem Compiler, dass es die entsprechende Klasse gibt und dass er das Auftreten des Klassennamens nicht als Fehler ansehen soll. Allerdings erfährt der Compiler durch eine Vorwärtsdeklaration nichts über die Details wie Methoden oder Propertys der Klasse – dazu ist eine Importanweisung notwendig. Vorwärtsdeklarationen sind jedoch sehr nützlich, um zyklische Imports zu vermeiden.

Hilfe bei der Speicherverwaltung

Xcode verändert nicht nur die Deklaration Ihrer Klasse, sondern auch deren Implementierung. Wenn Sie eine Outlet-Property anlegen, setzt Xcode in der Methode `viewDidUnload` die Property auf `nil`, und bei manuellem Referenzenzählen sendet es in `dealloc` außerdem noch ein `release` an das Attribut. Mit automatischem Referenzenzählen sollten Sie für Outlets den Speicherverwaltungstyp `weak` verwenden, sofern es sich bei dem verknüpften Objekt nicht um ein Element auf der obersten Ebene im NIB handelt (siehe dazu auch Abschnitt 3.2.11).

Anstelle von Outlet-Propertys können Sie auch Outlet-Attribute erzeugen. Dazu müssen Sie das Gummiband in den Attributblock, also zwischen die geschweiften Klammern, der Klassendeklaration ziehen. Allerdings halten sich Outlet-Attribute nicht an die Speicherverwaltungsregeln. Ein Outlet-Attribut bekommt ein `retain` gesendet, wenn es keinen Setter dafür gibt. Verwenden Sie also auch hier lieber Propertys als Attribute. Die neueren Xcode-Versionen legen allerdings keine Attributblöcke in den Header-Dateien mehr an, und Sie sollten darauf auch möglichst verzichten.

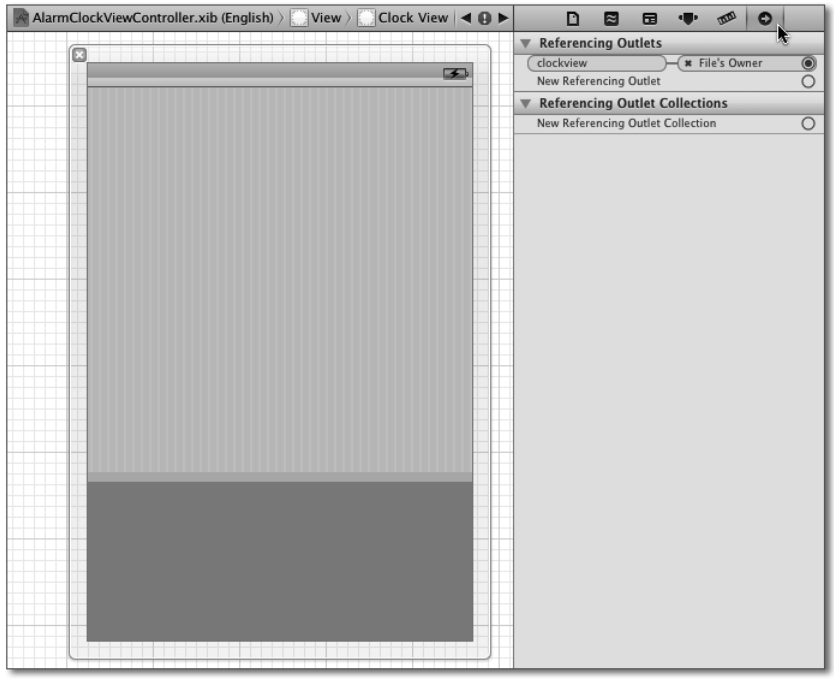

Abbildung 3.19 Der Verbindungsinspektor zeigt Verbindungen an.

Nach dem Laden der NIB-Datei verbindet Cocoa Touch diese Property mit dem View. Sie können sich die gesetzten und möglichen Verbindungen eines Objekts im Interface Builder über den Verbindungsinspektor ansehen (siehe Abbildung 3.19). Wenn Sie die Maus über eine Verbindung im Inspektor bewegen, hebt der Interface Builder den damit verbundenen View in der Ansicht hervor.

Indem Sie die entsprechenden Anweisungen in die Methoden `viewDidAppear:` und `viewWillDisappear:` einfügen, können Sie über den Viewcontroller jetzt eine Uhr starten. Die kompletten Methoden müssen Sie in die Implementierungsdatei *AlarmClockViewController.m* schreiben:

```
- (void)viewDidAppear:(BOOL)inAnimated {
    [super viewDidAppear:inAnimated];
    [self.clockView startAnimation];
}
- (void)viewWillDisappear:(BOOL)inAnimated {
    [self.clockView stopAnimation];
    [super viewWillDisappear:inAnimated];
}
```

Listing 3.18 Starten und Stoppen der Animation über den Viewcontroller

Außerdem sollten Sie die Headerdatei der Klasse `ClockView` in diese Implementierungsdatei einbinden, da Sie ansonsten Warnungen bei der Übersetzung erhalten. Dazu fügen Sie noch die Zeile

```
#import "ClockView.h"
```

an den Anfang der Implementierungsdatei ein. Hier reicht eine Vorwärtsdeklaration nicht aus, da der Controller ja die Methoden und Propertys der Klasse `ClockView` verwenden will. Setzen Sie diese Importanweisung unter die bereits enthaltene Importanweisung. Wenn Sie jetzt Ihr Programm im Simulator ausführen, bewegen sich die Zeiger einer Uhr.

3.2.2 Outlet-Collections

Fügen Sie zum Testen noch weitere Uhranzeigen in Ihren View (z. B. so wie im Beispielprojekt *Clock*) ein, die Sie später allerdings wieder entfernen sollten. Im laufenden Programm bewegen sich dann nur die Zeiger der Uhr, die Sie bereits über ein Outlet mit dem Viewcontroller verbunden haben. Falls Sie eine andere Uhr aktivieren möchten, können Sie einfach ein Gummiband von einem anderen Clockview auf die Outlet-Property ziehen; dann bewegen sich jedoch nur die Zeiger dieser Uhr. Das liegt daran, dass der Viewcontroller immer nur die Uhr startet, auf die er über das Outlet zugreifen kann.

Sie können jetzt für jede Uhr auch ein eigenes Outlet mit den entsprechenden Start- und Stoppanweisungen anlegen, um die Uhren analog zu starten und zu stoppen. Das ist jedoch sehr unelegant und sieht nicht sehr schön aus, da Sie mehrmals die gleichen Anweisungen schreiben müssen. Für das mehrfache Ausführen der gleichen Anweisung gibt es doch schließlich Schleifen.

Seit iOS 4 können Sie Outlet-Collections verwenden, um mehrere Objekte auf einmal zu verbinden. Sie erzeugen sie ähnlich wie Outlets, indem Sie eine Verbindung von einem Objekt im Interface Builder in die Headerdatei des Controllers ziehen. Lassen Sie die Maustaste los, und wählen Sie im Pop-Over-Dialog unter CONNECTION den

Punkt Outlet Collection aus. In das Textfeld schreiben Sie `clockViews`. Nach einem Klick auf den Connect-Button legt Xcode eine Property mit dem Typ `NSArray` an und fügt eine Markierung `IBOutletCollection(ClockView)` vor der Deklaration des Typs ein. Die Deklaration in der Headerdatei sieht so aus:

```
@property (retain, nonatomic) IBOutletCollection(ClockView) ➲
    NSArray *clockViews;
```

Außerdem hat Xcode die Implementierungsdatei verändert. Sie können jetzt der Outlet-Collection beliebig viele Clockviews zuweisen, indem Sie Gummibänder von den Views zu dieser Property-Deklaration ziehen. Der Makroparameter `ClockView` in der Deklaration schränkt im Interface Builder die möglichen Objekte für die Zuweisung auf die Klasse `ClockView` ein. Durch diesen Verbindungstyp können Sie jetzt zum Starten und Stoppen der Uhren eine Schleife verwenden:

```
- (void)viewDidAppear:(BOOL)inAnimated {
    [super viewDidAppear:inAnimated];
    for(ClockView *theView in self.clockViews) {
        [theView startAnimation];
    }
}
- (void)viewWillDisappear:(BOOL)inAnimated {
    for(ClockView *theView in self.clockViews) {
        [theView stopAnimation];
    }
    [super viewWillDisappear:inAnimated];
}
```

Listing 3.19 Starten und Stoppen der Uhren über eine Outlet-Collection

Der Verbindungsinspektor zeigt Ihnen die Outlet-Collections des Objekts in einer eigenen Rubrik an (siehe Abbildung 3.20), und Sie können einzelne Verbindungen durch einen Klick auf das x-Kreuzchen vor der Verbindung wieder lösen.

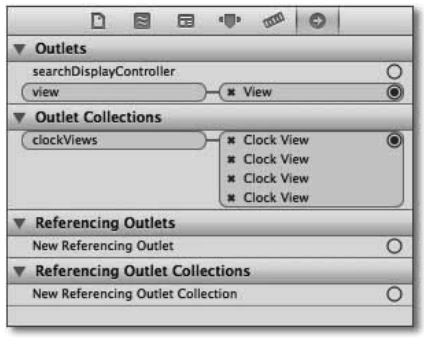

Abbildung 3.20 Outlet-Collections im Verbindungsinspektor

Achtung

Outlet-Collections verwenden zwar Arrays. Sie können jedoch die Reihenfolge der Elemente in dem Array nicht durch den Interface Builder festlegen.

3.2.3 Containerviews

Wenn Sie mit Ihrer App eine ältere iOS-Version als 4 unterstützen möchten oder die Reihenfolge der Views wichtig ist, können Sie statt der Outlet-Collection auch einen View als Container verwenden. Sie legen dazu alle Views, auf die Sie gesammelt zugreifen möchten, in einem gemeinsamen, gegebenenfalls nicht sichtbaren View an, und auf diesen View greifen Sie über ein herkömmliches Outlet zu. Häufig verwendet man für diese *Containerviews* ein `UIView` mit transparenter Hintergrundfarbe, der nur zur Beherbergung anderer Views dient. An die einzelnen Views gelangen Sie dann über die Property `subviews` des Containerviews. Allerdings kann dabei jeder View nur in maximal einem Containerview liegen.

Unterstützung älterer iOS-Versionen

Programmierer neigen dazu, immer die aktuelle Version des Betriebssystems vorauszusetzen. Bei den Endanwendern sieht das allerdings anders aus. Apple unterstützt iOS 4 und erst recht iOS 5 nicht oder nur eingeschränkt auf älteren iPhones und iPod touches. Besitzer eines iPhone 2G müssen mit iOS 3 vorliebnehmen, und iOS 5 läuft nicht mehr auf einem 3G. Mit zunehmender Vielfalt von Cocoa-Touch-Geräten ist damit zu rechnen, dass es auch immer eine nicht gerade kleine Benutzergruppe gibt, die nicht die aktuelle Version von iOS verwenden kann.

Überlegen Sie daher vor der Verwendung neuer Features, welche Reichweite Ihre App haben soll. Falls die App bezüglich der Hardware-Ressourcen sehr anspruchsvoll ist oder wenn sie Funktionen benötigt, die erst ab iOS 5 oder iOS 6 zur Verfügung stehen, ist die Unterstützung älterer Geräte nicht möglich. Bei einer App ohne besondere Hardware-Anforderungen lässt sich der Kreis der potenziellen Benutzer dagegen allein dadurch vergrößern, dass die App auf allen möglichen Geräten läuft – auch wenn das bedeutet, dass Sie nicht alle Annehmlichkeiten der neueren iOS-Versionen verwenden können.

In unserem Beispielprojekt existiert bereits ein View, den Sie als Containerview verwenden können. Es ist der Hauptview, in den Sie alle Zifferblätter gelegt haben. Für diesen Containerview brauchen Sie auch kein gesondertes Outlet, da Sie über die Property `view` des Viewcontrollers bereits auf diesen View zugreifen können. Sie können also die Outlet-Collection im letzten Listing durch die Property `subviews` der

`view`-Property des Viewcontrollers ersetzen. Sie müssen dazu nur die Zeilen mit den Schleifenköpfen durch

```
for(ClockView *theView in self.view.subviews) {
```

ersetzen. Das funktioniert allerdings nur, solange Sie keine Views einer anderen Klasse in den Hauptview legen.

3.2.4 Viewhierarchien

Die Reihenfolge der Subviews im darüber liegenden View ist entscheidend für die Anzeige. Sie können sich die Reihenfolge in der Baumdarstellung des Views ansehen, zu der Sie gelangen, indem Sie auf das Symbol am unteren Fensterrand von Xcode 4 klicken (siehe Mauszeiger in Abbildung 3.21).

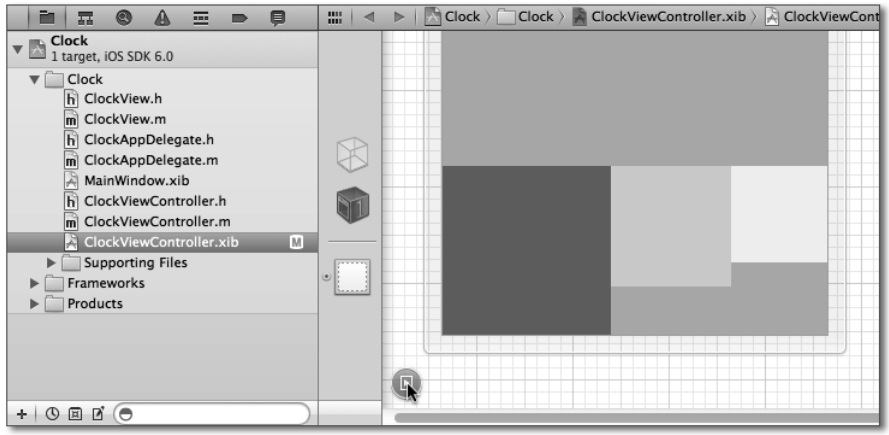

Abbildung 3.21 Umstellung auf Baumansicht (Mauszeiger)

In der Baumdarstellung können Sie sehen, wie die Views verschachtelt sind und in welcher Reihenfolge sie innerhalb einer Ebene liegen. Diese Anordnung wirkt sich auf die Darstellung und die Verarbeitung der Eingaben (oder auch *Touches*) aus. Je näher ein View zum Hauptview liegt, umso früher stellt Cocoa Touch ihn dar. Wenn also zwei Views einen gleichen Bereich auf dem Display verwenden, dann verdeckt der später dargestellte View den früher dargestellten.

In Abbildung 3.22 ist die Baumdarstellung eines Views zu sehen. Dabei haben die Views (❶ bis ❽) jeweils die gleichen Ziffern wie die Views in der Voransicht. Der rote View ❷ ist am nächsten zum Hauptview, weswegen ihn alle anderen Views verdecken. Der gelbe View ❻ liegt hinter dem grünen ❸ und verdeckt somit diesen, und der schwarze View ❼ liegt im gelben View, wodurch er diesen verdeckt.

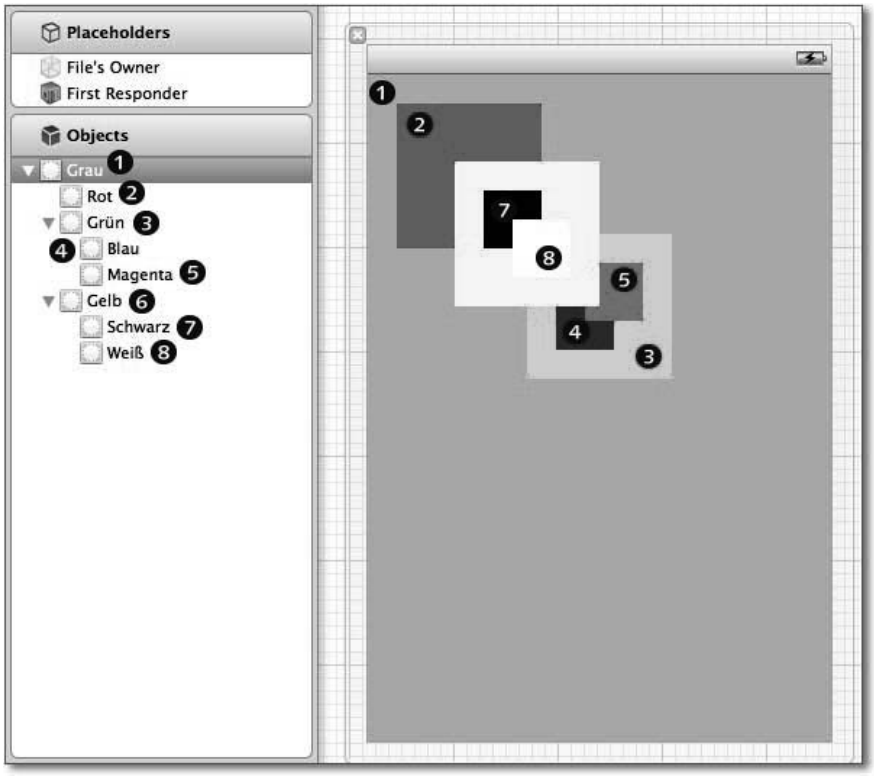

Abbildung 3.22 Baumdarstellung eines Views

Änderung der Viewhierarchie

Sie können auch in der Baumdarstellung die Anordnung der Views verändern. Ziehen Sie dazu die Views einfach an die gewünschte Position. Wenn Sie dort einen View auf einen anderen ziehen, fügt der Interface Builder den gezogenen View als letzten Subview des anderen Views ein.

Die Objekte auf der obersten Ebene nennt man auch *Top-Level-Objekte*. Das ist in Abbildung 3.22 der View mit dem Namen GRAU. Diese Objekte nehmen bezüglich der Speicherverwaltung – zumindest beim automatischen Referenzenzählen – eine Sonderstellung ein; dazu später mehr.

3.2.5 Actions

Bislang haben die verwendeten Views nur etwas auf dem Bildschirm dargestellt. Viele Views können indes auch Eingaben verarbeiten. Im Gegensatz zu den meisten Bildschirmen von Desktop-Computern ist bei iOS-Geräten das Ausgabegerät ja auch

gleichzeitig das Eingabegerät. Diese Views haben in der Regel die Klasse `UIControl` als Oberklasse und werden im Folgenden auch kurz als *Controls* bezeichnet. Typische Controls sind beispielsweise Buttons, Schieberegler und Textfelder.

Neben den Outlets können Sie auch Action-Verbindungen herstellen. Mit diesen Verbindungen verknüpfen Sie die von den Controls gesendeten Ereignisse mit einer Methode eines Controllers. Wenn Sie eine Action-Verbindung zu einem Control erzeugen, bekommt das Control ein Zielobjekt und eine Actionmethode übergeben. Dieses Vorgehen bezeichnet man auch als *Target-Action-Mechanismus*.

Actionmethoden müssen eine von drei vorgegebenen Signaturen haben und zumindest in der Deklaration für den Rückgabetyp das Makro `IBAction` verwenden. Dieses Makro wird während der Übersetzung durch den C-Typ `void` ersetzt. Es zeigt dem Interface Builder an, dass es sich bei einer Methode um eine Actionmethode handelt, zu der Actionverbindungen erlaubt sind.

Die drei möglichen Signaturen für Actionmethoden sind:

```
-(IBAction)methodenName;
-(IBAction)methodenName:(id)inSender;
-(IBAction)methodenName:(id)inSender ➲
               forEvent:(UIEvent *)inEvent;
```

Listing 3.20 Mögliche Signaturen für Actionmethoden

Den Methodennamen können Sie dabei natürlich frei wählen. Der erste Parameter in der zweiten und dritten Variante enthält das Control, das das Ereignis abgeschickt hat. Als Typ wurde hier `id` verwendet, der auf jede Klasse passt. Sie dürfen jedoch auch einen konkreten Typ, beispielsweise `UIButton *`, verwenden. Dann können Sie in der Methode auch die Punktnotation für dieses Control verwenden.

Über den Event-Parameter der dritten Variante können Sie zusätzliche Informationen über das auslösende Ereignis ermitteln. Darüber erhalten Sie beispielsweise die genauen Fingerpositionen zum Auslösezeitpunkt. Die liefert Ihnen der Event in Objekten der Klasse `UITouch`. Eine Möglichkeit, die Fingerposition zu bestimmen, stellt Listing 3.21 dar. Dabei bestimmt die Methode `locationInView:` die Fingerposition immer relativ zu dem angegebenen View. Das Listing berechnet sie also relativ zu dem Control, das das Ereignis ausgelöst hat.

```
- (IBAction)touchWithSender:(id)inSender
                   forEvent:(UIEvent *)inEvent {
   UITouch *theTouch = [inEvent.allTouches anyObject];
   CGPoint thePoint = [theTouch locationInView:inSender];
   // Touchposition in thePoint auswerten
}
```

Listing 3.21 Ermittlung der Touchposition

Sie sollen jetzt einen Schalter (das ist ein View der Klasse `UISwitch`) dafür verwenden, den Animationsstatus einer Uhr aus dem Clock-Projekt zu steuern. Fügen Sie dazu einen `UISwitch` hinzu. Das ist ein Schiebeschalter mit zwei Zuständen für »an« und »aus«. Nach dem Einfügen ziehen Sie von dem Schiebeschalter ein Band in den Header des Viewcontrollers. In dem Pop-over-Dialog haben Sie nun eine zusätzliche Auswahlmöglichkeit ACTION. Füllen Sie diesen Dialog wie in Abbildung 3.23 gezeigt aus.

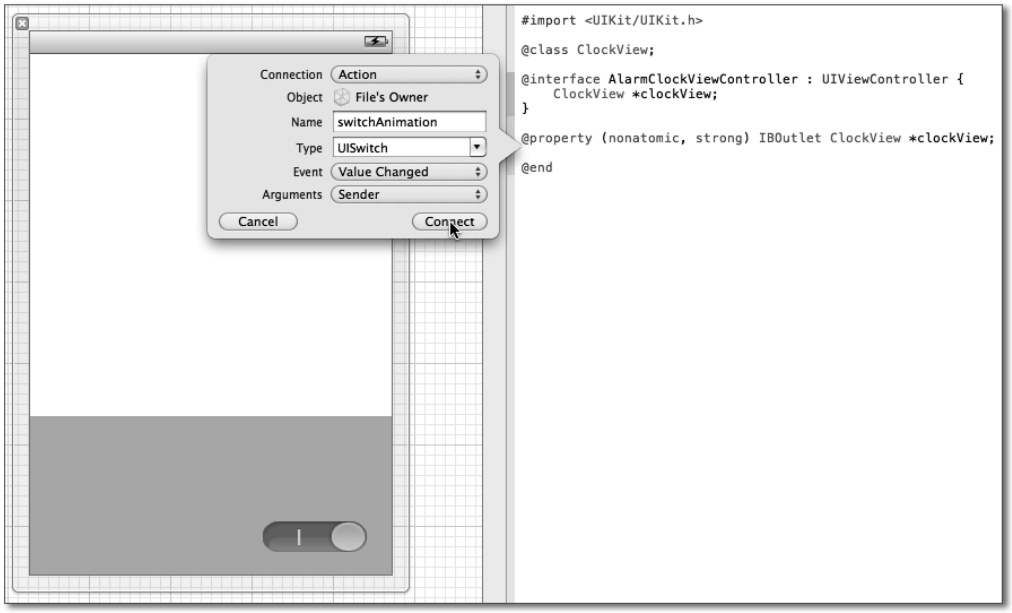

Abbildung 3.23 Anlegen einer Actionmethode

Das Control sendet das Ereignis `Value Changed` immer dann, wenn sich der Wert ändert, den das Control verwaltet. Bei einem `UISwitch` ist das ein boolescher Wert, der die Schalterstellung repräsentiert.

Sie können nun die Actionmethode wie folgt in der Datei *AlarmClockViewController.m* implementieren:

```
- (IBAction)switchAnimation:(UISwitch *)inSender {
    if(inSender.on) {
        [self.clockView startAnimation];
    }
    else {
        [self.clockView stopAnimation];
    }
}
```

Listing 3.22 Actionmethode zum Starten und Stoppen der Animation

Außerdem sollte die Methode `viewWillDisappear:` die Uhr stoppen. Wenn Sie nun das Projekt ausführen, können Sie die Uhr über den Schalter starten und stoppen. Allerdings steht nach dem Starten die Uhr, obwohl der Schalter auf »ein« steht. Sie können das umgehen, indem Sie entweder die Uhr in `viewDidAppear:` starten oder im Attributinspektor des Schalters dessen Anfangszustand auf »aus« ändern.

Eine weitere Alternative besteht darin, auch für den Schalter ein Outlet anzulegen. Die Methode `viewDidAppear:` kann dann den Wert des Schalters auslesen und in Abhängigkeit davon die Uhr starten. Der Vorteil dieser Lösung ist, dass Sie den Anfangszustand der Uhr über den Interface Builder, also ohne Veränderung des Programmcodes, einstellen können. Außerdem behält die Uhr ihren Animationszustand bei, wenn die App zwischenzeitlich einen anderen Viewcontroller anzeigt. Das ist bei diesem kleinen Projekt mit einem Viewcontroller vielleicht kein so großer Vorteil, kann allerdings bei sehr umfangreichen und komplexen Applikationen sehr angenehm sein. Wenn Sie die entsprechende Outlet-Property `animationSwitch` nennen, sieht der Code dafür so aus:

```
- (void)viewDidAppear:(BOOL)inAnimated {
    [super viewDidAppear:inAnimated];
    if(self.animationSwitch.on) {
        [clockView startAnimation];
    }
}
```

Listing 3.23 Starten der Animation in Abhängigkeit von einem Schalter

Alles für die Tonne

Denken Sie daran, die zusätzlichen Uhren wieder aus dem View zu entfernen. Sie brauchen sie für das Beispielprojekt nicht mehr. Ebenso brauchen Sie auch den Schalter nicht mehr. Sie können ihn also auch entfernen.

3.2.6 Ereignisse

Sie haben im Beispielprogramm das Ereignis `Value Changed` eines Schalters in einer Methode Ihres Viewcontrollers verarbeitet. Ein Control kann noch eine Reihe weiterer Ereignisse senden. Diese Ereignisse lassen sich in drei Kategorien unterteilen:

- Ereignisse für Wertänderungen
- Ereignisse für Gesten
- Ereignisse für Textänderungen

Ereignisse für Wertänderungen

Dieses Ereignis haben Sie bereits kennengelernt, und es hat den Typ `UIControlEvent-ValueChanged`. Es wird von Controls gesendet, die einen Wert manipulieren. Die Klassen, die dieses Ereignis versenden, finden Sie in Tabelle 3.1; die Spalte »Wert« enthält dabei den Namen der Methode, über die Sie den geänderten Wert abfragen können.

Klasse	Wert	Beschreibung
`UIDatePicker`	`date`	Erlaubt die Auswahl eines Datums, einer Uhrzeit oder von beidem auf einmal. Außerdem kann dieser View einen animierten Countdown anzeigen.
`UIPageControl`	`currentPage`	Erlaubt die Navigation zwischen verschiedenen Seiten über kleine Punkte wie im Springboard.
`UIRefreshControl`	`refreshing`	Stellt Tableviews eine Standardfunktionalität zum Erneuern der Zellinhalte bereit.
`UISegmentedControl`	`selectedSegmentIndex`	Stellt mehrere Buttons nebeneinander dar, von denen der Nutzer aber nur einen auswählen kann (Radiobutton-Leiste).
`UISlider`	`value`	Erlaubt die nahezu stufenlose Auswahl eines Wertes über einen Schieberegler.
`UISwitch`	`on`	Ein Schalter mit zwei stabilen Zuständen
`UIStepper`	`value`	Erlaubt die Auswahl eines Wertes über einen Plus- und einen Minusknopf. Dieses Control gibt es seit iOS 5.

Tabelle 3.1 Controls, die das Ereignis »Value Changed« senden

Ereignisse für Gesten

Ein wichtiger Erfolgsfaktor des iPhones ist dessen Gestensteuerung. Es gibt in iOS zwei Klassen von Gesten. Der Nutzer kann über *Berührungsgesten* Eingaben auf dem Bildschirm machen. Das können einfache Fingerdrücke oder komplexere Mehrfingerbewegungen auf dem Touchscreen sein. Dieser Abschnitt beschäftigt sich aus-

schließlich mit dieser Gestenklasse. Eine weitere grundsätzliche Möglichkeit, Eingaben vorzunehmen, sind Bewegungs- und Schüttelgesten, die wir in Kapitel 6, »Models, Layer, Animationen«, behandeln.

Berührungsgesten lösen eine Reihe von unterschiedlichen Ereignissen aus. Eine Berührungsgeste kann sich über mehrere Views erstrecken. Die Ereignisse werden hingegen immer an das Control gesendet, das das erste Ereignis einer Geste empfangen hat.

- `UIControlEventTouchDown` leitet alle Gesten ein und sendet das Control bei der ersten Berührung.
- `UIControlEventTouchDownRepeat` sendet das Control bei mehreren, schnell aufeinanderfolgenden Berührungen nach `UIControlEventTouchDown`. Sie können mit diesem Ereignistyp beispielsweise Doppel- oder Dreifachtaps auswerten. Die Anzahl der Taps können Sie über die Property `tapCount` aus den Touchobjekten des Ereignisses erfragen. Die Touchobjekte erhalten Sie über die Property `allTouches` der Klasse `UIEvent` (siehe Listing 3.21).
- `UIControlEventTouchUpInside` beendet eine Geste, wenn sich der Finger zuletzt innerhalb des auslösenden Controls befunden hat. Sie verwenden diesen Ereignistyp in der Regel als Aktionsauslöser bei Buttons.
- `UIControlEventTouchUpOutside` beendet eine Geste, wenn sich der Finger zuletzt außerhalb des auslösenden Controls befunden hat. Sie verwenden dieses Ereignis in der Regel, um den Abbruch einer Geste durch den Nutzer zu verarbeiten.
- `UIControlEventTouchCancel` bricht eine Geste ab. Dieser Ereignistyp wird beispielsweise an Controls innerhalb eines Scrollviews gesendet, wenn Cocoa Touch erkennt, dass Sie mit der Geste scrollen möchten und die Ereignisse nicht das Control betreffen. Cocoa Touch sendet dieses Ereignis auch, wenn es die Gestenverarbeitung wegen einer Speicherwarnung abbrechen muss.
- `UIControlEventTouchDragInside` zeigt eine Fingerbewegung innerhalb des Controls an.
- `UIControlEventTouchDragOutside` zeigt eine Fingerbewegung außerhalb des Controls an.
- `UIControlEventTouchDragExit` sendet das Control, wenn Sie die Finger aus dem Control heraus bewegen. Dieser Ereignistyp markiert den Übergang von `UIControlEventTouchDragInside` zu `UIControlEventTouchDragOutside`.
- `UIControlEventTouchDragEnter` sendet das Control beim Verschieben des Fingers in das Control hinein. Dieser Ereignistyp markiert den Übergang von `UIControlEventTouchDragOutside` zu `UIControlEventTouchDragInside`.

Sie können sich sehr komplexe Gesten ausdenken und mit diesen Ereignistypen umsetzen. Allerdings scheinen die von außerhalb des Controls gesendeten Ereig-

nisse nicht so zuverlässig zu funktionieren. Es kann beispielsweise vorkommen, dass Sie bei einer Fingerverschiebung aus dem Control heraus noch weitere `UIControlEventTouchDragInside`-Ereignisse empfangen. Die Umschaltung auf `UIControlEventTouchDragOutside` erfolgt teilweise erst, wenn sich der Finger schon lange außerhalb des Controls befindet. Wenn Sie also diese Ereignistypen verwenden möchten, sollten Sie ausgiebige Tests auf Geräten mit einplanen.

Viele Gesten lassen sich indes relativ einfach umsetzen, und Sie können beispielsweise Mehrfachtaps mit folgender Actionmethode auswerten, die Sie mit dem Ereignistyp `UIControlEventTouchDownRepeat` verbinden:

```
- (IBAction)handleMultiTap:(id)inSender➲
                  forEvent:(UIEvent *)inEvent {
    UITouch *theTouch = inEvent.allTouches.anyObject;
    if(theTouch.tabCount == 2) {
        // Doppel-Tap erkannt
    }
}
```

Listing 3.24 Erkennung einer Double-Tap-Geste

In den Body der `if`-Abfrage können Sie den Code einsetzen, den Sie bei einem Doppeltap ausführen wollen. Die Zahl in der Bedingung gibt die notwendige Anzahl der Taps an. Wenn Sie sie entsprechend anpassen, können Sie mit dem Code auch Dreifach- oder Vierfachtaps erkennen.

Jede Geste sendet mindestens zwei Ereignisse: `UIControlEventTouchDown` und einen Touch-Up-Event. Sie können bei Gesten für Fingerbewegungen `UIControlEventTouchDown` zur Initialisierung der Geste und `UIControlEventTouchUpInside` oder `-Outside` zum Beenden verwenden. Gegebenenfalls sollten Sie auch das Ereignis `UIControlEventTouchCancel` auswerten. Über dieses Ereignis können Sie Änderungen bei einem Gestenabbruch rückgängig machen.

Die Gestenverarbeitung für Ereignisse lässt sich jedoch nur sehr schlecht wiederverwenden. Das ist gerade bei komplexen Gesten ungünstig. Für die Gestenverarbeitung gibt es noch andere Möglichkeiten (siehe Abschnitt 3.2.8).

Ereignisse für Textänderungen

Diese Ereignisse unterstützt derzeit nur die Klasse `UITextField`, die ein Feld für die Eingabe eines einzeiligen Textes darstellt. Sie können zwar auch hier Actionmethoden mit `UIEvent`-Parametern verwenden. Für diesen Parameter bekommen Sie allerdings immer `nil` übergeben.

- `UIControlEventEditingDidBegin` markiert den Beginn der Eingabe in das Textfeld.
- `UIControlEventEditingDidChange` zeigt die Änderungen des Textes im Textfeld an.

- `UIControlEventEditingDidEnd` markiert das Ende der Eingabe in das Textfeld.
- `UIControlEventEditingDidEndOnExit` markiert ebenfalls das Ende der Eingabe in das Textfeld. Das Textfeld sendet dieses Ereignis anscheinend nur, wenn der Nutzer das Textfeld durch Drücken des Return-Knopfes über die Systemfunktion beendet. Leider geht die Apple-Dokumentation auf dieses Ereignis nicht genauer ein.

Ein View, dessen Klasse keine Unterklasse von `UIControl` ist, kann keine Ereignisse verschicken. Wenn Sie indes ein Ereignis senden möchten, beispielsweise um die Berührung eines Bildes zu verarbeiten, können Sie das Bild einfach in einen View mit der Klasse `UIControl` legen. Sie legen ein solches Control wie einen Clockview an. Ziehen Sie ein Viewobjekt an die gewünschte Stelle, und ändern Sie dessen Klasse im Identitätsinspektor auf `UIControl`. Im Verbindungsinspektor dieses Views finden Sie dann alle beschriebenen Ereignisse. Sie können dagegen auch einen Button verwenden, der ebenfalls die Anzeige von Bildern unterstützt.

3.2.7 Controlzustände und Buttons

Neben den Ereignissen unterstützen Controls auch drei boolesche Systemzustände. Diese Zustände werden durch eine Bitmaske in der Property `state` des Controls abgebildet.

- `UIControlStateHighlighted` ist aktiv, solange der Nutzer das Control gedrückt hält. Diesen Zustand können Sie auch über die Property `highlighted` abfragen oder setzen.
- `UIControlStateDisabled` setzt das Control inaktiv. Es verarbeitet keine Eingaben und versendet auch keine Ereignisse. Diesen Zustand können Sie auch über die Property `enabled` abfragen oder setzen. Der Wert der Property ist dabei jedoch genau umgekehrt zu dem Wert in der Bitmaske.
- `UIControlStateSelected` zeigt den Auswahlzustand des Controls an. Diesen Zustand können Sie auch über die Property `selected` abfragen oder setzen.

Ein Control kann mehrere Zustände gleichzeitig aktiviert haben. Wenn kein Zustand in der Bitmaske gesetzt ist, hat das Control den Zustand `UIControlStateNormal`. Am ausgiebigsten machen Buttons von den Zuständen Gebrauch.

Buttons sind relativ komplexe Controls, die mehrere Darstellungselemente unterstützen. Die Darstellungselemente können Sie in Abhängigkeit von den Zuständen setzen. Sie können für jede Zustandskombination eine eigene Darstellung festlegen. Außerdem können Sie über die Property `adjustsImageWhenHighlighted` zusätzlich festlegen, dass der Button beim Drücken noch einen Glüheffekt anzeigt. Abbildung 3.24 stellt rechts jeweils einen Button im Zustand `highlighted` ohne und mit Glüheffekt dar.

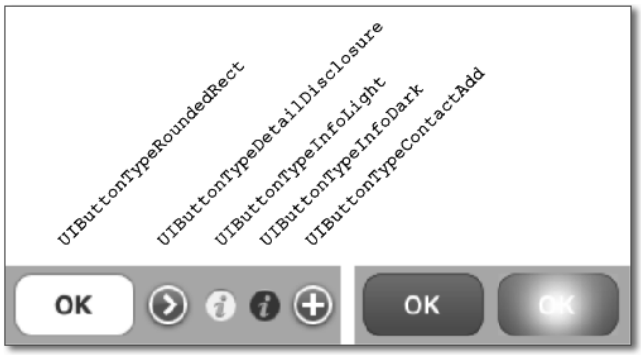

Abbildung 3.24 Die verschiedenen Darstellungstypen für Buttons

Ein Button kann einen von sechs vordefinierten Darstellungstypen haben, wobei fünf Typen ein festes Aussehen haben (siehe Abbildung 3.24, links) und Sie beim sechsten Typ, `UIButtonTypeCustom`, das Aussehen selbst bestimmen können.

Ein *Custom-Button* unterstützt drei Darstellungselemente: ein Vorder- und ein Hintergrundbild sowie einen Titel (siehe Abbildung 3.25). Für den Titel können Sie außerdem noch die Text- und die Schattenfarbe zustandsabhängig setzen.

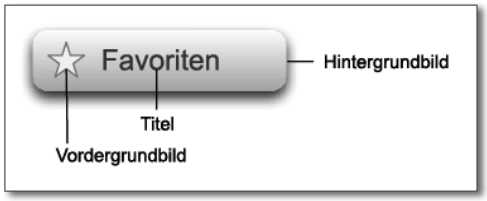

Abbildung 3.25 Die Darstellungselemente eines Buttons

Sie können also über diese fünf Eigenschaften das wesentliche Aussehen des Buttons festlegen, wobei Sie das Vordergrundbild und den Titel über *Insets* beliebig positionieren können. Das können Sie entweder über die Propertys `imageEdgeInsets` beziehungsweise `titleEdgeInsets` oder über den Interface Builder (siehe Abbildung 3.26, unterer Bereich) machen. Über die Property `contentEdgeInsets` können Sie außerdem den kompletten Inhalt – also Bild und Titel – auf einmal positionieren.

Wenn Sie für eine Zustandskombination eine Darstellungseigenschaft nicht definieren, wird stattdessen die des Normalzustands verwendet. Im Interface Builder können Sie alle Eigenschaften für die einfachen Zustände, aber nicht für Kombinationen aus mehreren Zuständen festlegen. Sie können also beispielsweise das Hintergrundbild für den Zustand `selected` festlegen hingegen nicht für `selected + highlighted`. Das geht nur über Programmcode.

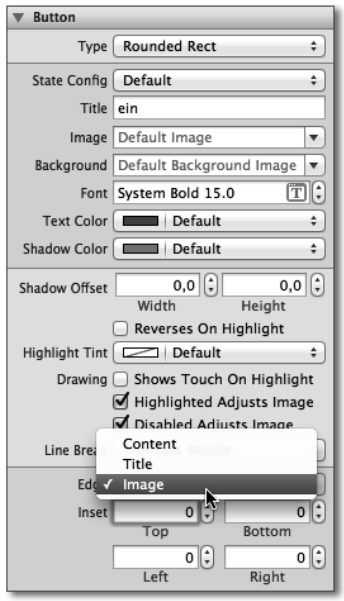

Abbildung 3.26 Der Attributinspektor eines Buttons

Durch die Möglichkeit, für die verschiedenen Buttonzustände unterschiedliche Darstellungen festzulegen, können Sie Ihren Apps relativ einfach optisch ansprechende Buttons hinzufügen und damit auch fehlende Buttontypen (beispielsweise Checkboxen) basteln; so lässt sich die Uhr auch über einen einrastenden Button anstatt eines Schalters steuern. Das Einrasten erreichen Sie durch die Invertierung des Zustands `selected`. Die Actionmethode `switchAnimation:` in der Datei *AlarmClockViewController.m* müssen Sie dazu wie folgt ändern:

```
- (IBAction)switchAnimation:(UIButton *)inSender {
    inSender.selected = !inSender.selected;
    if(inSender.selected) {
        [self.clockView startAnimation];
    }
    else {
        [self.clockView stopAnimation];
    }
}
```

Listing 3.25 Actionmethode für einen einrastenden Button

Natürlich müssen Sie auch die Deklaration der Methode in der Headerdatei entsprechend ändern. Legen Sie im Interface Builder einen Button an, und stellen Sie über dessen Attributinspektor für die Zustände DEFAULT, HIGHLIGHTED und SELECTED die Titel *ein, klick* beziehungsweise *aus* ein.

Vorsicht bei geänderten Methodensignaturen

Wenn Sie die Signatur einer Actionmethode im Programmcode ändern, sollten Sie alle Actionverbindungen im Interface Builder zu der ursprünglichen Methode löschen. Ansonsten erhalten Sie einen Laufzeitfehler. Besonders tückisch sind dabei die Änderungen zwischen Signaturen ohne und mit einem Parameter, da sie sich nur durch einen Doppelpunkt unterscheiden. Zwischen den Signaturen `switchAnimation` und `switchAnimation:` besteht ein Unterschied!

Verbinden Sie den Button mit der Actionmethode, indem Sie ein Band vom Button zum File's Owner ziehen. Wenn Sie das Band dort loslassen, erscheint ein Dialog, in dem Sie die Actionmethode auswählen können. Im Verbindungsinspektor des Buttons können Sie sehen, dass der Interface Builder automatisch die Actionmethode mit dem Ereignis *Touch Up Inside* verbunden hat. Der Button sendet dieses Ereignis, wenn der Finger den Button losgelassen hat.

Nach dem Start des Programms ist die Uhr ausgeschaltet und der Button zeigt die Beschriftung EIN. Wenn Sie den Button drücken, wechselt die Beschriftung auf KLICK und nach dem Loslassen auf AUS. In diesem Zustand verbleibt der Button, und die Uhr läuft.

Durch erneutes Drücken des Buttons wechselt die Beschriftung wieder auf EIN. Der Button hat doch den Zustand `highlighted`; warum erscheint hier nicht auch die Beschriftung KLICK? Das liegt daran, dass in diesem Fall die Zustände `highlighted` und `selected` gesetzt sind. Für diese Zustandskombination haben Sie jedoch keinen Titel festgelegt, sodass der Button den Titel des Zustands `default` verwendet.

Den Titel für diese Zustandskombination können Sie, wie bereits erwähnt, nicht im Interface Builder festlegen, sondern müssen das im Programmcode machen. Dafür legen Sie ein Outlet auf den Button an. Die Klasse `UIViewController` besitzt eine weitere Methode, `viewDidLoad`, die Cocoa Touch aufruft, nachdem der View geladen wurde. In dieser Methode können Sie die Werte für diese Zustandskombination festlegen.

Da der Button für den Zustand `selected + highlighted` den gleichen Titel wie für den Zustand `selected` verwenden soll, kopiert die Methode den entsprechenden Titel aus dem Button. Dadurch können Sie den Titel allein über den Interface Builder pflegen. Durch diese Änderung zeigt der Button in der oben beschriebenen Situation jetzt auch den Titel KLICK anstatt EIN an.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    NSString *theTitle = [self.switchButton ➲
        titleForState:UIControlStateHighlighted];
```

```
    [self.switchButton setTitle:theTitle forState:➲
        UIControlStateSelected | UIControlStateHighlighted];
}
```

Listing 3.26 Initialisierung eines Buttons nach dem Laden des Views

3.2.8 Direkte Gestenverarbeitung

Wie bereits erwähnt, können Sie den Target-Action-Mechanismus für die Gestenverarbeitung einsetzen. Dieses Vorgehen ist indes recht aufwendig, wenn Sie eine Geste wiederverwenden möchten. Cocoa Touch bietet noch andere Möglichkeiten, um Gesten zu verarbeiten.

Damit ein View die Touch-Ereignisse empfängt, müssen Sie immer dessen Property `userInteractionEnabled` setzen. Cocoa Touch durchsucht in der View-Hierarchie nur diese Views und sendet die Ereignisse an den tiefsten View, unter dem der Berührungspunkt des Fingers liegt. Mit »der tiefste View« ist gemeint, dass es in diesem View keinen weiteren Subview an der Fingerposition mit gesetzter Property `userInteractionEnabled` gibt.

Der View muss auch wollen

Sie können über `userInteractionEnabled` die Touchverarbeitung komplett ausschalten. Wenn Sie diesen Property-Wert also auf `NO` setzen, reagieren weder der View noch dessen Subviews mehr auf Berührungen. Dabei ist es egal, ob Sie ein Control verwenden oder die Touches direkt oder über eine andere Methodik auswerten. Bei `userInteractionEnabled = NO` ignoriert der View Ihre Berührungen – mögen Sie ihn auch noch so zärtlich streicheln.

Die Klasse `UIView` stellt zwei Methoden bereit, um den Subview zu ermitteln, der die Gesten verarbeitet. Die Methode `hitTest:withEvent:` bestimmt zu einem Punkt den Subview, der die Gestenverarbeitung übernehmen soll. Sie verwendet dazu die Methode `pointInside:withEvent:`, die entscheidet, ob der angegebene Punkt in dem View liegt oder nicht. Während der Eventparameter bei beiden Methoden `nil` sein darf, müssen Sie den Punkt relativ zu dem Koordinatensystem des Views angeben. Wenn Sie diese Methode aus dem Programmcode aufrufen, müssen Sie also in der Regel den Punkt erst umrechnen.

Die Klasse `UIView` stellt dafür die Methoden `convertPoint:fromView:` und `convertPoint:toView:` bereit. Die erste Methode rechnet den Punkt aus dem Koordinatensystem des zweiten Parameters in das Koordinatensystem des Nachrichtenempfängers um. Bei der zweiten Methode sind die Koordinatensysteme des zweiten Parameters und des Empfängers bei der Berechnung vertauscht.

... und raus bist du!

Wenn Sie die View-Bestimmung für die Gestenverarbeitung anpassen möchten, sollten Sie in Ihren Views die Methode `pointInside:withEvent:` überschreiben. Ein Überschreiben der Methode `hitTest:withEvent:` ist in der Regel nicht notwendig.

Sofern Sie den Property-Wert `clipsToBounds` eines Views auf `NO` setzen, kann jeder Subview des Views über dessen Grenzen hinaus zeichnen. Allerdings empfängt der Subview nur da Touches, wo er auch innerhalb seiner umgebenden Views liegt. Diese Situation stellt Abbildung 3.27 dar: Cocoa Touch stellt den Subview, der in der Hierarchie im Superview liegt, zwar komplett auf dem Bildschirm dar; dieser empfängt jedoch nur Ereignisse für die Gesten in dem gekennzeichneten, berührungsempfindlichen Bereich. Die anderen Berührungen erhält in diesem Beispiel der Hauptview.

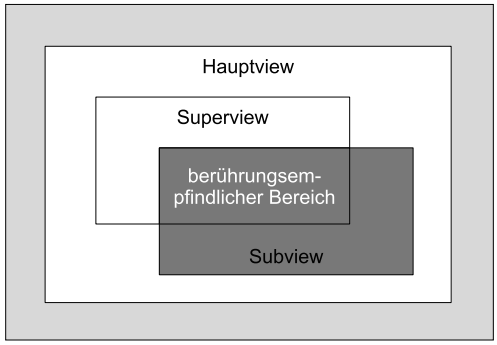

Abbildung 3.27 Berührungsempfindlicher Bereich eines Subviews

Gestenverarbeitung über Responder

Die Klasse `UIResponder` ist die direkte Oberklasse von `UIView` und `UIViewController`. Sie stellt mehrere Methoden zur Auswertung von Gesten bereit. Die Methoden für die Berührungsgesten haben jeweils zwei Parameter und liefern `void` zurück. Der erste Parameter ist eine Menge der Klasse `NSSet` von `UITouch`-Objekten, die eine Fingerberührung auf dem Bildschirm repräsentieren. Der zweite Parameter ist ein `UIEvent`-Objekt.

Der Methodenaufruf von `touchesBegan:withEvent:` leitet eine Gestensequenz ein. Die folgenden Fingerbewegungen erhält der Responder über die Methode `touchesMoved:withEvent:`. Abgeschlossen wird die Geste über einen Aufruf von entweder `touchesEnded:withEvent:` oder `touchesCancelled:withEvent:`. Dabei kennzeichnet ein Aufruf der Methode `touchesCancelled:withEvent:`, dass das System eine Geste abbrechen möchte (siehe dazu die Beschreibung zu `UIControlEventTouchCancel` in Abschnitt 3.2.6).

Diese Methoden sind die Grundlagen der Gestenverarbeitung. Allerdings führt die Verwendung dieser Methoden in der Regel zu einer relativ festen Koppelung von View und Controller, da der View meistens das Ergebnis einer Geste an den Controller weiterreichen muss. Falls Sie dennoch eine eigene Gestenverarbeitung über diese Responder-Methoden realisieren möchten, sollten Sie die Kommunikation mit dem Controller über Delegation realisieren. Eine direkte Verwendung von Controller-Eigenschaften im View ist eine Verletzung des MVC-Architekturmusters, weil dadurch eine Abhängigkeit des Views von der Controllerschicht entsteht.

Responder-Methoden und Delegation

In Kapitel 2, »Grundlagen«, haben Sie bereits die Delegation als Muster kennengelernt. Um den View mit der Controllerschicht lose zu koppeln, erstellen Sie ein Protokoll, das die Delegate-Methoden deklariert. Außerdem spendieren Sie dem View eine Property, um ihn mit dem Delegate verknüpfen zu können. Dann können die Responder-Methoden des Views immer eine Delegate-Methode aufrufen, wenn sie die gewünschte Geste erkannt haben. Sie sollten also nicht die Events einfach weiterleiten, sondern lieber die erkannten Gesten.

Angenommen, Ihr View soll Kreisbewegungen erkennen, dann könnte das Protokoll die Delegate-Methode `circleView:didDetectsCircleWithCenter:radius:` bereitstellen, die der View nach der Erkennung eines Kreises aufruft. Wenn Sie diese Geste aber in unterschiedlichen Views erkennen möchten, sollten Sie dafür lieber einen *Gesture Recognizer* einsetzen.

Gestenverarbeitung über Controls

Eine andere Möglichkeit, einen Controller lose an einen View zu koppeln, ist die Verwendung des Target-Action-Mechanismus über eigene Unterklassen von `UIControl`. Diese Klasse stellt ebenfalls vier Methoden zur Gestenverarbeitung bereit, die den Methoden der Klasse `UIResponder` sehr ähneln. Ihre Namen lauten `beginTrackingWithTouch:withEvent:`, `continueTrackingWithTouch:withEvent:` und `endTrackingWithTouch:withEvent:`, und sie erhalten jeweils zwei Parameter, wovon der erste ein Touchobjekt und der zweite der Event ist. Im Gegensatz zu den Responder-Methoden erhalten Sie hier also nicht alle Berührungen des Bildschirms als direkte Parameter. Falls Sie eine Mehrfingergeste implementieren möchten, können Sie jedoch alle Berührungen über die Methode `allTouches` des Events abfragen. Außerdem gibt es noch die Methode `cancelTrackingWithEvent:`, die das Control analog zu `touchesCancelled:withEvent:` beim Abbruch einer Geste aufruft.

Über die Gestenverarbeitung soll aus der Uhr ein Wecker entstehen. Bei einem analogen Wecker gibt es neben den Zeigern für die Zeitanzeige einen weiteren Zeiger für die Alarmzeit. In der Wecker-App soll der Nutzer diesen Zeiger durch Berührung und

Bewegung mit dem Finger einstellen können. Das Control muss also eine Drehbewegung mit einem Finger auf dem Bildschirm erkennen. Diese Bewegung ähnelt dem Wählen auf einem Telefon mit Wählscheibe. Das Control soll dabei auch die Darstellung des Alarmzeigers übernehmen. Legen Sie zunächst eine neue Klasse `ClockControl` als Unterklasse von `UIControl` an.

Das Control erhält für die Darstellung der Alarmzeit die Property `time` mit dem Typ `NSTimeInterval`, der eine Zeitdauer in Sekunden als Fließkommawert enthält. Sie können die Alarmzeit über die Methode `angle` in einen Winkel umrechnen und entsprechend über die Methode `setAngle:` als Winkelwert setzen. Dabei sind 12 Stunden genau

$12 \times 60 \times 60 = 43200$ Sekunden

und die entsprechen einem Winkel von 360°, und damit können Sie aus der Zeit t über

$2t\pi / 43200 = t\pi / 21600 = w$

den Winkel w berechnen. Wenn Sie die letzte Gleichung mit $21600 / \pi$ multiplizieren, erhalten Sie die Gleichung

$t = 21600w / \pi$

um die Zeit aus dem Winkel zu berechnen. Diese beiden Gleichungen bilden die Grundlage für die Implementierung der Methoden `angle` und `setAngle:`

```
- (CGFloat)angle {
    return self.time * M_PI / 21600.0;
}

- (void)setAngle:(CGFloat)inAngle {
    self.time = 21600.0 * inAngle / M_PI;
}
```

Listing 3.27 Umrechnung der Alarmzeit in einen Winkel und umgekehrt

Da auch das Control die Methoden `midPoint` und `pointWithRadius:angle:` benötigt, lagern Sie sie aus der Klasse `ClockView` in die Kategorie `UIView(AlarmClock)` aus. Rufen Sie dazu den Menüpunkt File • New • File... auf, und wählen Sie die Vorlage Objective-C class extension in der Rubrik iOS • Cocoa Touch aus. Wenn Sie im folgenden Dialog »AlarmClock« unter Extension Name und »UIView« unter Class eingeben (siehe Abbildung 3.28), legt Xcode beim Speichern die Dateien *UIView+AlarmClock.h* und *UIView+AlarmClock.m* an. In die Headerdatei tragen Sie die Deklarationen für die beiden Methoden ein. Außerdem sollten Sie die Systemheaderdatei für das *UIKit* importieren. Die komplette Datei finden Sie in Listing 3.28.

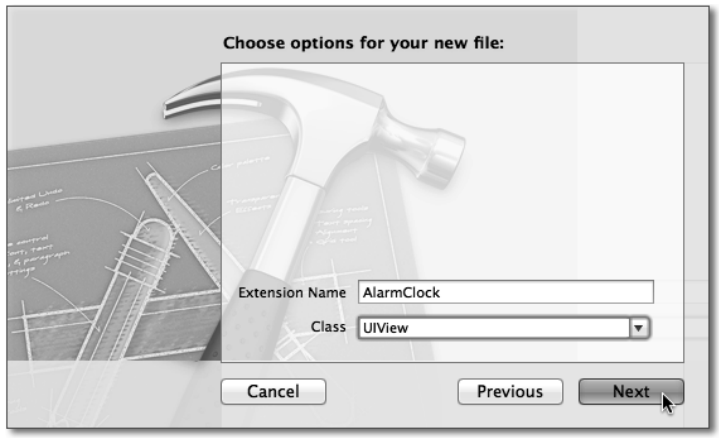

Abbildung 3.28 Anlegen einer Kategorie für die Wecker-App

```
#import <UIKit/UIKit.h>

@interface UIView (AlarmClock)

- (CGPoint)midPoint;
- (CGPoint)pointWithRadius:(CGFloat)inRadius➲
    angle:(CGFloat)inAngle;

@end
```

Listing 3.28 Deklaration der Kategorie

Analog verschieben Sie die Implementierungen dieser beiden Methoden aus der Klasse `ClockView` in den Implementierungsblock der Kategorie in der Datei *UIView+AlarmClock.m*. Durch das Entfernen der Methoden zeigt Xcode in der Klasseninplementierung Fehler an. Um sie zu eliminieren, sollten Sie die Kategorie in die Datei *ClockView.m* importieren. Dazu fügen Sie die Anweisung `#import "UIView+AlarmClock.h"` vor dem Implementierungsblock ein. Damit Sie die Methoden auch im neuen Control verwenden können, fügen Sie die Importanweisung außerdem an den Anfang von *ClockControl.m* ein.

Die Methode `angleWithPoint:` berechnet über die C-Funktion `atan2` zu einem Punkt im Control den Winkel der Linie, die vom Mittelpunkt zu diesem Punkt führt:

```
- (CGFloat)angleWithPoint:(CGPoint)inPoint {
    CGPoint theCenter = [self midPoint];
    CGFloat theX = inPoint.x - theCenter.x;
    CGFloat theY = inPoint.y - theCenter.y;
```

```
    CGFloat theAngle = atan2f(theX, -theY);
    return theAngle < 0 ? theAngle + 2.0 * M_PI : theAngle;
}
```

Listing 3.29 Berechnung des Winkels zu einem Punkt

Der Winkel, den die Funktion `atan2` berechnet, liegt zwischen $-\pi$ und π. Negative Werte müssen Sie also auf entsprechende positive Werte umrechnen. Abbildung 3.29 stellt diese beiden Fälle grafisch dar. Dabei befinden sich positive Werte auf der rechten und negative auf der linken Kreishälfte.

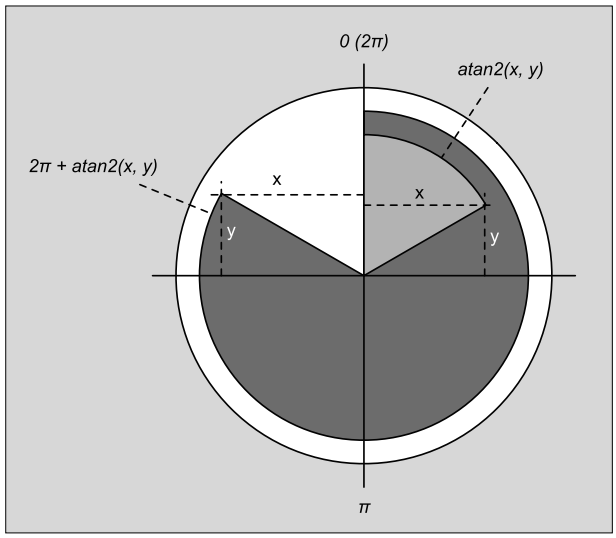

Abbildung 3.29 Umrechnung von Punkten in Winkel

Der Zeiger soll sich jedoch nur verstellen lassen, wenn die Fingerberührung in der Nähe des Zeigers stattfindet. Das können Sie am einfachsten erreichen, wenn Sie die Methode `pointInside:withEvent:` überschreiben:

```
- (BOOL)pointInside:(CGPoint)inPoint
          withEvent:(UIEvent *)inEvent {
    CGFloat theAngle = [self angleWithPoint:inPoint];
    CGFloat theDelta = fabs(theAngle - self.angle);
    return theDelta < 2 * M_PI / 180.0;
}
```

Listing 3.30 Prüfung, ob ein Punkt in der Nähe des Alarmzeigers liegt

Die Variable `theDelta` enthält den Winkelabstand des Fingers vom Zeiger. Wenn dieser Winkel kleiner als 2° ist, soll das Control den Touch verarbeiten. Mit diesen Methoden können Sie nun die Gestenverabeitung implementieren. Der Inhalt der

drei Methoden `beginTrackingWithTouch:withEvent:`, `continueTrackingWithTouch:withEvent:` und `endTrackingWithTouch:withEvent:` ist dabei fast gleich, sodass das Control für diesen Code sinnvollerweise die Hilfsmethode `updateAngleWithTouch:` bereitstellt. Sie ermitteln jeweils aus der Fingerposition den Punkt im Control und rechnen diesen über die Methode `angleWithPoint:` in den entsprechenden Winkel um. Diesen Winkel übergeben Sie an die Methode `setAngle:`, die daraus die entsprechende Weckzeit berechnet. Nach der Neuberechnung müssen die Methoden noch den Zeiger für die Weckzeit durch einen Aufruf von `setNeedsDisplay` neu zeichnen lassen. Die letzte Anweisung der Methode versendet schließlich das Value-Changed-Ereignis. Sie ruft also bei den entsprechenden Targets des Controls die jeweilige Actionmethode auf.

```
- (void)updateAngleWithTouch:(UITouch *)inTouch {
    CGPoint thePoint = [inTouch locationInView:self];
    self.angle = [self angleWithPoint:thePoint];
    [self setNeedsDisplay];
    [self sendActionsForControlEvents:➲
        UIControlEventValueChanged];
}
```

Listing 3.31 Aktualisierung der Daten in den Tracking-Methoden

Die Methode `cancelTrackingWithEvent:` sollte den Ausgangszustand des Controls bei Gestenabbruch wiederherstellen. Dazu muss es den aktuellen Winkel zu Beginn der Gestenverarbeitung speichern und in dieser Methode wieder an `setAngle:` übergeben. Für die Speicherung verwendet das Control eine private Property `savedAngle`. Die Implementierung dieser Methoden sieht also wie folgt aus:

```
- (BOOL)beginTrackingWithTouch:(UITouch *)inTouch➲
    withEvent:(UIEvent *)inEvent {
    self.savedAngle = self.angle;
    [self updateAngleWithTouch:inTouch];
    return YES;
}

- (BOOL)continueTrackingWithTouch:(UITouch *)inTouch➲
    withEvent:(UIEvent *)inEvent {
    [self updateAngleWithTouch:inTouch];
    return YES;
}

- (void)endTrackingWithTouch:(UITouch *)inTouch➲
    withEvent:(UIEvent *)inEvent {
```

```
    [self updateAngleWithTouch:inTouch];
}

- (void)cancelTrackingWithEvent:(UIEvent *)inEvent {
    self.angle = self.savedAngle;
    [self setNeedsDisplay];
}
```

Listing 3.32 Die Methoden für die Gestenverarbeitung

Jetzt fehlt dem Control nur noch die Implementierung der Methode `drawRect:`, die einfach nur einen blauen Strich als Zeiger zeichnet. Dabei verwendet sie während der Gestenverarbeitung einen transparenten Blauton als Rückmeldung für den Nutzer. Diesen Zustand können Sie über die Property `tracking` der Klasse `UIControl` erfragen.

```
- (void)drawRect:(CGRect)inRectangle {
    CGContextRef theContext = UIGraphicsGetCurrentContext();
    CGRect theBounds = self.bounds;
    CGPoint theCenter = [self midPoint];
    CGFloat theRadius = CGRectGetWidth(theBounds) / 2.0;
    CGPoint thePoint = [self pointWithRadius:theRadius * 0.7 ➲
        angle:self.time * M_PI / 21600.0];

    CGContextSaveGState(theContext);
    CGContextSetRGBStrokeColor(theContext,➲
        0.0, 0.0, 1.0, self.tracking ? 0.5 : 1.0);
    CGContextSetLineWidth(theContext, 7.0);
    CGContextSetLineCap(theContext, kCGLineCapRound);
    CGContextMoveToPoint(theContext, ➲
        theCenter.x, theCenter.y);
    CGContextAddLineToPoint(theContext, ➲
       thePoint.x, thePoint.y);
    CGContextStrokePath(theContext);
    CGContextRestoreGState(theContext);
}
```

Listing 3.33 Zeichnen des Alarmzeigers

In der XIB-Datei können Sie jetzt das Control in den Clockview legen, wenn Sie dem Control eine transparente Hintergrundfarbe geben. Ziehen Sie dazu zuerst ein `UIView`-Objekt aus der Bibliothek auf das Ziffernblatt, und weisen Sie ihm die Klasse `ClockControl` über dessen Identitätsinspektor zu. Das Control sollte die gleiche Größe wie der Clockview haben. Außerdem müssen Sie dessen Property `userInteractionEnabled` im Attributinspektor auf `YES` setzen, damit Cocoa Touch die Gesten auch an das Control sendet.

Um Ihr neues Control zu testen, legen Sie ein Label in den View und verbinden es über ein Outlet mit dem Namen `timeLabel` mit dem Viewcontroller. In diesem Label soll der Wecker die Alarmzeit als Text anzeigen. Außerdem verbinden Sie das Control im Interface Builder über den Ereignistyp *Value Changed* mit der Methode `updateTimeLabel` des Viewcontrollers, deren Implementierung Sie in Listing 3.34 finden. Sobald Sie den Alarmzeiger bewegen, sendet das Control ja dieses Ereignis. Die Methode schreibt die aktuelle Weckzeit in das Label. Die Variable `theTime` enthält dabei die Alarmzeit in Minuten. Die Stunden und Minuten für die Anzeige berechnen Sie daraus durch eine Division durch 60, wobei der Divisionsrest der Minutenanteil ist.

```
- (IBAction)updateTimeLabel {
    NSInteger theTime = round(self.clockControl.time / 60.0);
    NSInteger theMinutes = theTime % 60;
    NSInteger theHours = theTime / 60;

    self.timeLabel.text = [NSString stringWithFormat:➲
        @"%d:%02d", theHours, theMinutes];
}
```

Listing 3.34 Erzeugung einer Zeichenkette aus einer Alarmzeit

Wenn Sie den Alarmzeiger in der App bewegen, können Sie sehen, dass der Controller auch die angezeigte Alarmzeit kontinuierlich aktualisiert. Dem Wecker fehlt jetzt nur noch die Weckfunktion, also das Klingeln. Außerdem muss der Viewcontroller beim Loslassen des Zeigers den Alarm konfigurieren. Diese beiden Themen behandelt Abschnitt 3.3, und die Implementierung der Weckfunktion geschieht über die Actionmethode `updateAlarm`, die Sie über das Ereignis *Touch Up Inside* mit dem Control verbinden.

Mehrere Ereignistypen behandeln

Der Viewcontroller verarbeitet also zwei unterschiedliche Ereignistypen des Controls. Das ist gerade bei Controls, die kontinuierliche Werte liefern, sinnvoll. Der Ereignistyp *Value Changed* dient zur Aktualisierung des Views und somit zur interaktiven Rückmeldung des Zustands an den Nutzer, während das Ereignis *Touch Up Inside* die Verarbeitung des endgültigen Wertes in der App auslöst. Sie können jedoch auch noch andere Ereignisarten sinnvoll verwenden. Beispielsweise können Sie durch den Ereignistyp *Touch Up Outside* einen Abbruch der Operation ermöglichen, indem Sie dafür eine Actionmethode implementieren, die den alten Zustand wiederherstellt.

Gestenverarbeitung über Gesture Recognizer

Mit der iOS-Version 3.2[5] beziehungsweise 4.0 hat Apple eine weitere Möglichkeit der Gestenverarbeitung eingeführt. Die Klasse `UIGestureRecognizer` ermöglicht eine Implementierung von Gestensteuerungen, die von den Views getrennt ist. Damit können Sie jede beliebige Gestenverarbeitung zu jedem View hinzufügen. Der Gesture Recognizer trennt also die Eingabeverarbeitung von der Darstellung des Views. Dadurch sind diese beiden Aufgaben (Darstellung und Ereignisverarbeitung) vollkommen unabhängig voneinander oder auch *orthogonal* zueinander, und Sie können die entsprechenden Komponenten (Views und Gesture Recognizer) nahezu beliebig miteinander kombinieren.

Die Klasse `UIView` bietet zwei Methoden und eine Property zum Verwalten von Gesture Recognizern an. Mit `addGestureRecognizer:` fügen Sie einem View einen neuen Gesture Recognizer hinzu, den Sie mit `removeGestureRecognizer:` wieder entfernen können. Mit der Property `gestureRecognizers` können Sie alle Gesture Recognizers des Views abfragen oder auf einmal setzen. Mit Xcode 4.2 können Sie die Gesture Recognizer auch direkt über die Bibliothek (siehe Abbildung 3.30) des Interface Builders zu Ihren Views hinzufügen, indem Sie sie auf die Views ziehen.

Abbildung 3.30 Die Gesture Recognizer in Xcode 4.2

Die Klassen sind in der gleichen Reihenfolge wie die Icons in Abbildung 3.30 aufgeführt.

1. `UITapGestureRecognizer` erkennt einzelne oder mehrere Berührungen des haltenden Views. Sie können sowohl die notwendige Tap-Zahl als auch die Anzahl der beteiligten Finger zur Auslösung vorgeben.
2. `UIPinchGestureRecognizer` erkennt die Bewegung von zwei Fingern aufeinander zu oder voneinander weg. Diese Geste wird meistens zum Heraus- beziehungsweise Hereinzoomen verwendet.
3. `UIRotationGestureRecognizer` erkennt die Bewegung von zwei sich umeinander drehenden Fingern, wie sie viele Apps für die Drehung von Bildern eingesetzen.
4. `UISwipeGestureRecognizer` erkennt Wischbewegungen. Sie können die Anzahl der beteiligten Finger angeben, jedoch damit nur die Richtung und keine Distanzen auswerten.
5. `UIPanGestureRecognizer` erkennt Verschiebe-Bewegungen in einem View. Sie können damit beispielsweise das Verschieben eines Views innerhalb eines anderen

5 Diese Version war auf das iPad beschränkt. Für alle anderen Gerätetypen gibt es die Gesture Recognizer erst ab iOS 4.0.

durchführen (*Dragging*), da Sie mit diesem Recognizer auch die Richtung und die Distanz der Geste ermitteln können. Auch bei diesem Gesture Recognizer können Sie die Anzahl der beteiligten Finger vorgeben.

6. `UILongPressGestureRecognizer` erkennt längere Berührungen des zugrunde liegenden Views. Diese Gestenauswertung eignet sich sehr gut, wenn ein View unterschiedliche Gestenarten unterstützen soll. Beispielsweise können Sie eine Karte scrollen und durch längeres Drücken eine Markierung setzen. Auch bei diesem Gesture Recognizer können Sie die Anzahl der beteiligten Finger einschränken. Außerdem können Sie eine minimale Dauer für die Berührungslänge wählen.

Sie können auch eigene Unterklassen der Klasse `UIGestureRecognizer` schreiben und so eine eigene Gestenverarbeitung implementieren. Dafür besitzt die Klasse die gleichen vier Methoden, die auch die Klasse `UIResponder` bereitstellt. Zusätzlich gibt es eine Methode `reset`. Sie können diese Methode überschreiben, um nach dem Abschluss einer Geste Ihren Gesture Recognizer für die nächste Geste zurückzusetzen.

Gesture Recognizer unterstützen wie Controls den Target-Action-Mechanismus – allerdings ohne Ereignistypen. Sie können über die Methode `addTarget:action:` eine neue Aktion hinzufügen, die Sie mit `removeTarget:action:` auch wieder entfernen können.

Das Beispielprojekt verwendet einen `UILongPressGestureRecognizer`, um die Alarmzeit zusätzlich zur Zeigerdrehung einstellen zu können. Eine Berührung des Zifferblatts, die länger als vier Zehntelsekunden dauert, schaltet den Alarmzeiger ein und setzt ihn an die Position des Fingers. Legen Sie einen *Longpress Recognizer* an, indem Sie das entsprechende Objekt aus der Bibliothek auf das Clockview-Objekt in der Baumansicht ziehen.

Das Ein- und Ausschalten des Alarmzeigers und des Labels erfolgt über die Property `alarmHidden`, die auch die Sichtbarkeit dieser Views steuert. Sie ändert außerdem noch den Zustand eines Schalters über das Outlet `alarmSwitch`, mit dem der Nutzer den Alarm explizit ein- oder ausschalten kann.

```
- (BOOL)alarmHidden {
    return self.clockControl.hidden;
}

- (void)setAlarmHidden:(BOOL)inAlarmHidden {
    self.alarmSwitch.on = !inAlarmHidden;
    self.clockControl.hidden = inAlarmHidden;
    self.timeLabel.hidden = inAlarmHidden;
}
```

Listing 3.35 Sichtbarkeit der Alarmanzeige steuern

Die Zuordnung ist wichtig

Der Gesture Recognizer funktioniert nicht richtig, wenn Sie ihn nicht zu dem View, sondern zu dem Control hinzufügen. Sie sollen ihn ja auch bei ausgeschaltetem Alarm benutzen können. Da Sie in diesem Zustand allerdings das Control versteckt haben, beachtet Cocoa Touch dann auch den Recognizer nicht. Das Ziffernblatt ist hingegen immer sichtbar. Sie können die Zuordnung des Recognizers übrigens in dessen Verbindungsinspektor überprüfen.

Die Actionmethode eines Gesture Recognizers hat entweder keinen oder genau einen Parameter. In diesem Parameter bekommen Sie den Gesture Recognizer übergeben, und er sollte dessen Typ haben. Die Actionmethode für den Wecker muss die Position des Fingers auslesen. Dazu bietet die Klasse `UIGestureRecognizer` die Methode `locationInView:`, mit der Sie die Fingerposition relativ zu dem Koordinatensystem eines Views berechnen können. Die Actionmethode `updateClockHand:` können Sie jetzt folgendermaßen implementieren:

```
- (IBAction)updateAlarmHand:↩
    (UIGestureRecognizer *)inRecognizer {
    CGPoint thePoint =↩
        [inRecognizer locationInView:self.clockControl];
    CGFloat theAngle =↩
        [self.clockControl angleWithPoint:thePoint];
    self.clockControl.angle = theAngle;
    [self.clockControl setNeedsDisplay];
    self.alarmHidden = NO;
    [self updateTimeLabel];
    if(inRecognizer.state == UIGestureRecognizerStateEnded) {
        [self updateAlarm];
    }
}
```

Listing 3.36 Actionmethode zur Aktualisierung des Alarmzeigers

Die Methode `updateAlarm` am Ende von Listing 3.36 sorgt dafür, dass der Wecker zur eingestellten Zeit auch klingelt. Wie das genau funktioniert, erfahren Sie, wie bereits erwähnt, in Abschnitt 3.3, »Lokale Benachrichtigungen«. Wenn Sie den Gesture Recognizer durch eine Berührung auslösen, ruft er die Actionmethode so lange auf, bis Sie den Finger wieder vom Bildschirm nehmen. Er registriert also auch alle Fingerbewegungen. Die App braucht indes die Auslösung des Alarmtons nicht jedes Mal zu aktualisieren. Eine Aktualisierung reicht aus, wenn der Nutzer den Zeiger loslässt.

Diese Situation können Sie über die Property `state` des Gesture Recognizers feststellen. Wenn sie den Wert `UIGestureRecognizerStateEnded` hat, hat der Nutzer die Geste beendet.

Die Qual der Wahl

Sie haben drei Möglichkeiten zur Gestenverarbeitung kennengelernt. Welche ist denn nun die beste? Das hängt natürlich von dem Anwendungsfall ab. Tabelle 3.2 listet die verschiedenen Möglichkeiten für die Gestenverarbeitung auf.

Gestenverarbeitung	Vor- und Nachteile
Überschreiben der Responder-Methoden	Vorteile 1. Die Gestenverarbeitung auch im Viewcontroller und nicht nur im View möglich. 2. Auswertung mehrerer unterschiedlicher Gesten innerhalb eines Views
	Nachteile 1. Gefahr der Verletzung des MVC-Musters: Abhängigkeit des Views vom Controller 2. Die Gestenverarbeitung ist mit der Viewklasse fest verbunden, was die Wiederverwendung sehr stark einschränkt.
Unterklasse von `UIControl`	Vorteile 1. Nutzung des Target-Action-Mechanismus und dadurch keine Abhängigkeit des Views vom Viewcontroller 2. Nutzung der Action-Connections im Interface Builder 3. Hohe Wiederverwendbarkeit möglich
	Nachteile 1. Keine eigenen Eventtypen
Unterklasse von `UIGestureRecognizer`	Vorteile 1. Gestenverarbeitung komplett von der Darstellung getrennt 2. Viele fertige Recognizer vorhanden 3. Höchster Wiederverwendungsgrad, da die Gestenerkennung komplett von der Darstellung getrennt ist

Tabelle 3.2 Vor- und Nachteile verschiedener Möglichkeiten zur Gestenerkennung

Der Hauptnachteil bei der Verwendung der Responder-Methoden (die enge Kopplung des Views an den Viewcontroller) kann natürlich auch bei undurchdachter Implementierung der anderen Verfahren auftreten. Bei der Verwendung des Responder-Mechanismus tappen jedoch viele Programmierer in diese Falle. Sie können eine starke Koppelung jedoch – wie beschrieben – über Delegation vermeiden.

3.2.9 Übergänge

Mit iOS 5 und den Storyboards hat Apple auch *Segues* (Übergänge) eingeführt. Diese Objekte beschreiben einen Übergang von einem Viewcontroller zu einem anderen. Übergänge haben somit als Verbindungstyp im Interface Builder einen sehr eingeschränkten Anwendungsbereich. Andererseits können Sie darüber schon im Interface Builder eine Verknüpfung zwischen den UI-Elementen und den Views festlegen. Deswegen nennt Apple diese neuen Beschreibungsdateien für die Views auch *Storyboards*.

Um die Funktionsweise der Übergänge zu veranschaulichen, erstellen Sie ein kleines Projekt. Legen Sie dazu in Xcode ein neues iPad-Projekt des Typs Single View Application an, und geben Sie an, dass dieses Projekt ein Storyboard und ARC verwenden soll. Öffnen Sie das Storyboard, und legen Sie einen neuen Button in den dort angezeigten View. Danach ziehen Sie einen neuen Viewcontroller (siehe Abbildung 3.31) auf die Zeichenfläche des Storyboards.

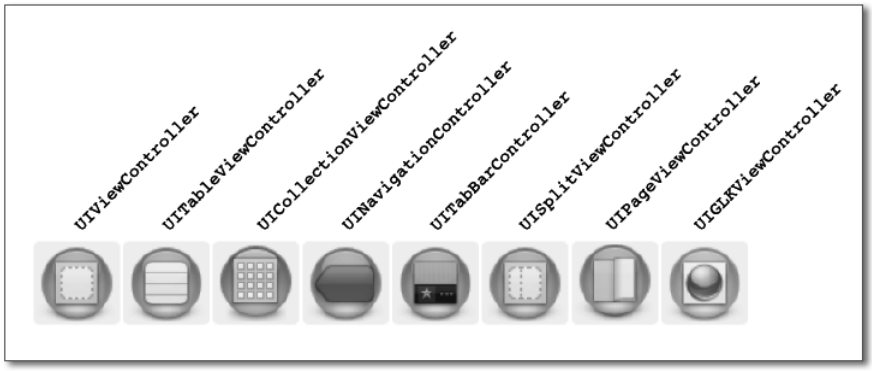

Abbildung 3.31 Die Viewcontroller im Interface Builder

Als Nächstes legen Sie einen Übergang an, indem Sie bei gedrückter rechter Maustaste ein Gummiband vom Button zu dem neuen Viewcontroller ziehen. Nachdem Sie in dem schwarzen Dialog den Eintrag `performSegueWithIdentifier:sender:` ausgewählt haben, zeigt der Interface Builder zwischen den Viewcontrollern einen grauen Pfeil mit einem Symbol in der Mitte an. Im Attributinspektor dieses Übergangs können Sie jetzt festlegen, wie der Wechsel erfolgen soll. Wählen Sie unter

Style die Option Modal und unter Presentation den Punkt Form Sheet aus (siehe Abbildung 3.32).

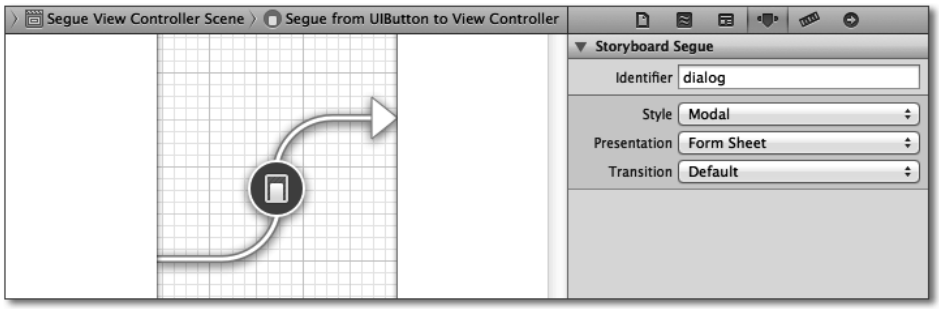

Abbildung 3.32 Ein Übergang zwischen zwei Viewcontrollern

Starten Sie das Projekt im Simulator, und klicken Sie den Button. Der Simulator graut die weiße Fläche aus und schiebt von unten ein weißes Rechteck auf den Bildschirm. Dieses Rechteck ist der neue Viewcontroller, den Sie im Storyboard angelegt haben.

Mit dieser Konfiguration zeigt die App ihn als modalen Dialog an. In der Methode `prepareForSegue:sender:` können Sie den neu erzeugten Viewcontroller konfigurieren. Der Übergang ruft diese Methode bei dem Ursprungsviewcontroller auf, bevor er den Zielviewcontroller anzeigt. Wenn Sie mehrere Übergänge von einem Viewcontroller aus aufrufen können, können Sie diese anhand ihrer Kennung unterscheiden. Die Kennung können Sie über das Eingabefeld Identifier im Attributinspektor des Übergangs setzen (siehe Abbildung 3.32). Die Unterscheidung im Programmcode kann dann anhand der Property `identifier` erfolgen, die die Kennung des Übergangs enthält. Listing 3.37 gibt Ihnen ein Beispiel dafür.

```
#import "SegueViewController.h"

@implementation SegueViewController

- (void)prepareForSegue:(UIStoryboardSegue *)inSegue➲
    sender:(id)inSender {
    if([inSegue.identifier isEqualToString:@"dialog"]) {
        id theDialog = inSegue.destinationViewController;
        // theDialog initialisieren
    }
}
@end
```

Listing 3.37 Initialisierung eines Controllers bei einem Übergang

Sie können Übergänge auch im Code über die Methode `performSegueWithIdentifier:sender:` auslösen. Mit dieser Methode lassen sich also Übergänge auch über Actionmethoden aufrufen:

```
- (IBAction)performSegue:(id)inSender {
    [self performSegueWithIdentifier:@"dialog"
        sender:inSender];
}
```

Listing 3.38 Auslösung eines Übergangs

3.2.10 Der Lebenszyklus eines Viewcontrollers

Der Viewcontroller verwaltet den View; er lädt ihn aus einer NIB-Datei oder einem Storyboard. Wie erzeugt Cocoa Touch hingegen den Viewcontroller? Auf diese Frage gibt es zwei Antworten. Entweder liegt er als serialisiertes Objekt in einer XIB-Datei beziehungsweise einem Storyboard, oder das Programm erzeugt ihn wie andere Objekte auch über eine direkte Anweisung im Code.

Für den initialen Viewcontroller der Wecker-App kommt, wenn Sie Xcode 4.2 oder neuer verwenden, die letzte Variante zum Einsatz. Wenn Sie die Implementierung der Klasse `AlarmClockAppDelegate` öffnen, finden Sie in der Methode `application:didFinishLaunchingWithOptions:` den entsprechenden Code. Die zweite Anweisung in der Methode – das ist die Zuweisung zu der Property `viewController` – enthält die Erzeugung des Viewcontrollers.

```
- (BOOL)application:(UIApplication *)inApplication➲
    didFinishLaunchingWithOptions:(NSDictionary *)inOptions {
    self.window = [[[UIWindow alloc] initWithFrame:➲
        [[UIScreen mainScreen] bounds]] autorelease];
    self.viewController = ➲
        [[[AlarmClockViewController alloc]➲
            initWithNibName:@"AlarmClockViewController"➲
            bundle:nil] autorelease];
    self.window.rootViewController = self.viewController;
    [self.window makeKeyAndVisible];
    return YES;
}
```

Listing 3.39 Erzeugung des Viewcontrollers in der Wecker-App

Ältere Xcode-Versionen haben dagegen eine weitere XIB-Datei mit dem Namen *MainWindow.xib* angelegt. Über diese Datei legt Xcode das Application-Delegate, den Viewcontroller und das Fenster der App an. Sie finden diese Objekte jeweils als Top-Level-Objekte in der XIB-Datei.

Sie können Ihre Viewcontroller und auch Objekte von beliebigen Klassen in NIB-Dateien anlegen. Dazu ziehen Sie das entsprechende Controllerobjekt (siehe Abbildung 3.31) einfach in die linke Seitenleiste des Interface Builders und verbinden es über ein Outlet mit dem File's Owner. Wenn Sie Objekte mit anderen Klassen anlegen wollen, verwenden Sie stattdessen den orangenen Würfel (siehe Abbildung 3.33). Sie können so Objekte mit nahezu beliebigen Klassen in der XIB-Datei oder im Storyboard anlegen.

Abbildung 3.33 Bibliothekssymbol für beliebige Objekte

Wenn Sie diese Objekte nach der Erzeugung initialisieren möchten, sollte deren Klasse eine Initialisierungsmethode `init` oder `initWithCoder:` besitzen, wobei Cocoa Touch Letztere bevorzugt. Alternativ können Sie auch den Initialisierungscode in die Methode `awakeFromNib` schreiben (siehe Abbildung 3.15). Zu beliebigen Objekten in NIB-Dateien folgen später noch Beispiele.

Tipp

Wenn Sie einen einfachen Viewcontroller über eine NIB-Datei erzeugen, sollten Sie unbedingt auf zwei Dinge achten. Erstens müssen Sie im Identitätsinspektor die Klasse des Viewcontrollers setzen. Andernfalls erzeugt Ihre App für den Viewcontroller ein Objekt mit der Klasse `UIViewController`. Zweitens sollten Sie im Attributinspektor des Controllers die richtige XIB-Datei mit dem passenden View auswählen.

Auch bei beliebigen Objekten müssen Sie deren Klasse im Identitätsinspektor setzen, ansonsten legt Ihnen Cocoa Touch ein schnödes `NSObject` an. Übrigens können Sie in diesen Objekten auch Outlets und Actionmethoden anlegen, die Sie mit Ihren Views verknüpfen können.

Sie haben bereits einige Methoden kennengelernt, die Cocoa Touch zu bestimmten Zeitpunkten im Leben eines Viewcontrollers aufruft. Wenn Sie beispielsweise die Methode `viewDidAppear:` überschreiben, können Sie eigenen Programmcode ausführen, unmittelbar nachdem Ihre App den View des Viewcontrollers angezeigt hat. Cocoa Touch benachrichtigt Sie also über ein bestimmtes Ereignis, das Ihren Viewcontroller betrifft. In Abbildung 3.34 ist der komplette Lebenszyklus eines Viewcontrollers anhand seiner Initialisierungs- und Benachrichtigungsmethoden sowie der Property `view` dargestellt. Bis auf die Property `view` dürfen Sie allerdings keine dieser Methoden direkt aufrufen. Sie dürfen sie nur überschreiben.

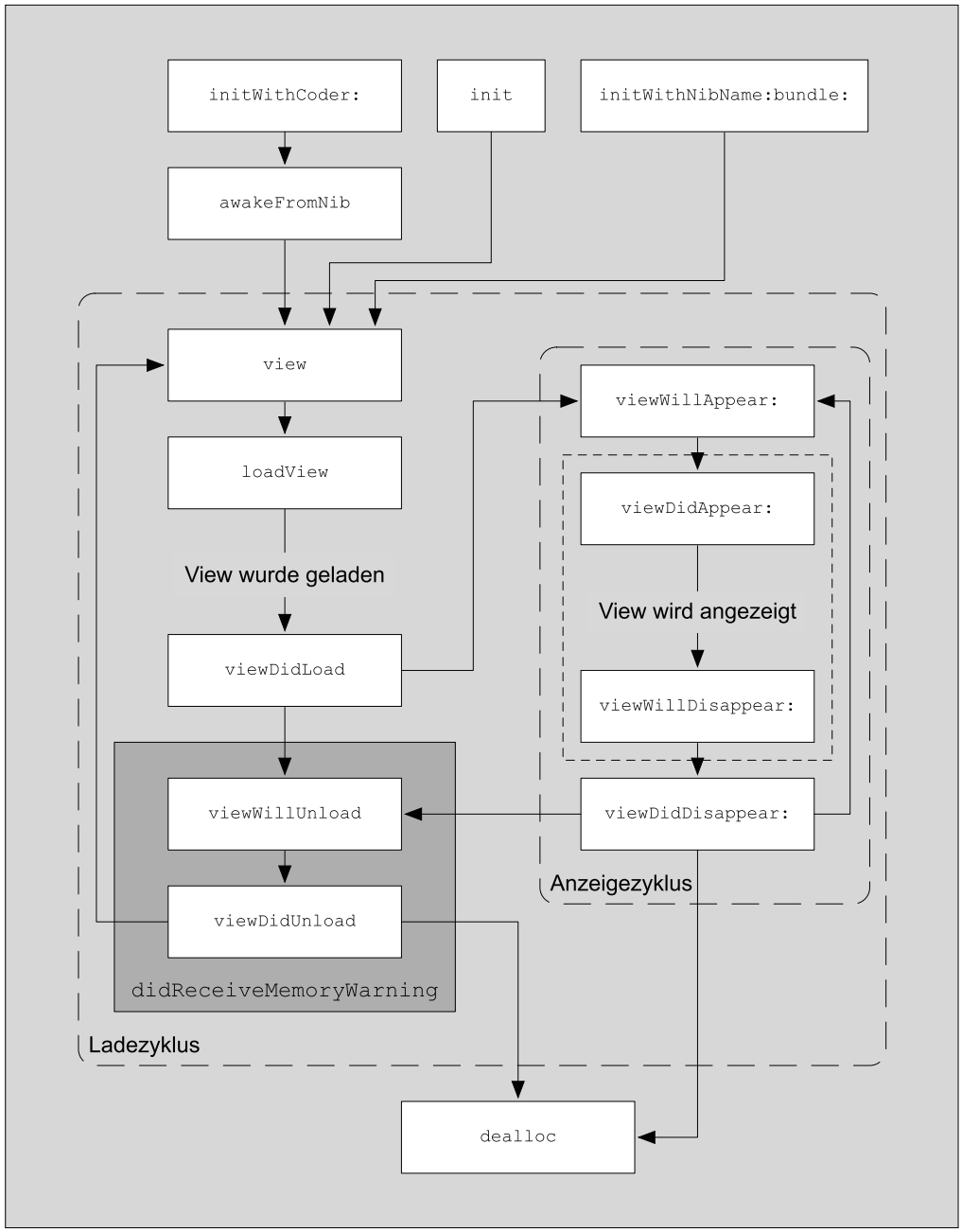

Abbildung 3.34 Der Lebenszyklus eines Viewcontrollers

Richtig überschreiben

Sie dürfen alle Methoden aus Abbildung 3.34 überschreiben. Dabei sollten Sie jedoch in der Methode – bis auf `view`, `setView:` und `loadView`[6] – auch immer die Methode in der Oberklasse aufrufen. Bei den drei genannten Methoden können Sie die Methode der Oberklasse aufrufen, müssen es jedoch nicht tun.

Bei initialisierenden Methoden wie `awakeFromNib` oder `viewDidAppear:` sollte der Aufruf immer am Anfang der Methode erfolgen. Bei den finalisierenden Methoden wie `viewWillDisappear:` oder `viewDidUnload` erfolgt der Aufruf der Supermethode hingegen erst am Schluss der überschreibenden Methode. Für die Methoden `viewDidAppear:` und `viewWillDisappear:` sieht das beispielsweise so aus:

```
- (void)viewDidAppear:(BOOL)inAnimated {
    [super viewDidAppear:inAnimated];
    // Hier steht Ihr Code
}
- (void)viewWillDisappear:(BOOL)inAnimated {
    // Hier steht Ihr Code
    [super viewWillDisappear:inAnimated];
}
```

Seit iOS 6 hat Apple die Methoden `viewWillUnload` und `viewDidUnload` als veraltet markiert. Sie dürfen sie zwar weiterhin verwenden, allerdings kann Apple die Unterstützung dafür mit einer zukünftigen iOS-Version ganz einstellen. Abschnitt 3.2.11, »Speicher- und Ressourcenverwaltung des Viewcontrollers«, geht noch genauer darauf ein, wie Sie die Verwendung dieser Methoden umgehen können.

Die Methoden `initWithCoder:` und `awakeFromNib` kennen Sie ja bereits von den View-Klassen. Diese Methoden kommen zum Zuge, wenn Sie den Viewcontroller in einer NIB-Datei oder einem Storyboard erzeugen.

Sie können einen Viewcontroller auch aus Ihrem Programmcode erzeugen. Dazu dienen die Methoden `init` und `initWithNibName:bundle:`. Wenn Sie `init` verwenden, müssen Sie entweder die Methode `loadView` implementieren, oder die NIB-Datei des Viewcontrollers muss bis auf die Dateiendung genauso wie die Klasse Ihres Viewcontrollers heißen. Wenn sie allerdings einen anderslautenden Namen haben soll, sollten Sie stattdessen die Initialisierungsmethode `initWithNibName:bundle:` verwenden, wobei Sie für den Bundle-Parameter `nil` verwenden können.

Das Überschreiben der Methode `awakeFromNib` ist in vielen Fällen sinnlos, da der Viewcontroller zum Zeitpunkt ihres Aufrufs den View noch nicht geladen hat und somit die Outlets noch nicht initialisiert sind. Es sind nur die Outlets gesetzt, die in

6 Bei ARC gehört hier natürlich auch `dealloc` dazu.

die NIB-Datei verweisen, die den Viewcontroller erzeugt. Ein Beispiel dazu gibt Abbildung 3.35, die die Hierarchien von zwei fiktiven Viewcontrollern enthält. Dabei lädt das Objekt VIEW CONTROLLER auf der linken Seite seinen View aus dem NIB auf der rechten Seite. Der Eigentümer (FILE'S OWNER) der rechten NIB-Datei ist ein Platzhalter für das Viewcontroller-Objekt aus dem linken NIB. Der Viewcontroller verlinkt also über Outlets sowohl auf Objekte aus dem linken als auch aus dem rechten NIB.

Wenn Cocoa Touch nun die Methode `awakeFromNib` des Viewcontrollers aufruft, bestehen nur die Outlets zu den beiden Objekten NAVIGATION CONTROLLER und OBJECT aus dem linken NIB, da er das rechte NIB ja noch nicht geladen hat. Die Verbindung zum View und die Outlets zum Label und dem Imageview hat Cocoa Touch hingegen noch nicht hergestellt. Diese Verbindungen kommen erst durch den Aufruf der `loadView`-Methode zustande.

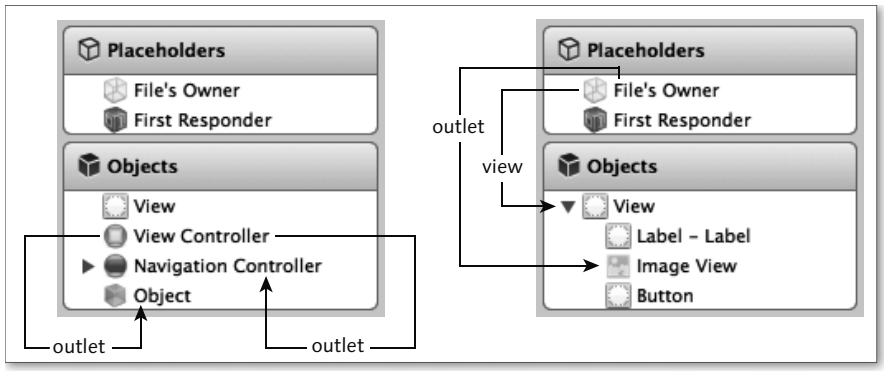

Abbildung 3.35 Outlets zwischen XIB-Dateien

Regeln für Outlets

Auf das Objekt eines Outlets eines Viewcontrollers können Sie

- frühestens in `viewDidLoad` zugreifen, wenn Sie das Outlet vom FILE'S OWNER zum Objekt gezogen haben.
- schon in `awakeFromNib` zugreifen, wenn die Verbindung von dem Viewcontroller in der Rubrik OBJECTS ausgeht.

Der Viewcontroller lädt den View beim ersten Zugriff auf die Property `view`. Das geschieht vor der ersten Anzeige des Views. Sie können allerdings auch das Laden erzwingen, indem Sie die Property auslesen. Sie können dafür Ihrem Viewcontroller beispielsweise die folgende Methode hinzufügen:

```
- (BOOL)forceLoadView {
    return self.view != nil;
}
```

Listing 3.40 Methode zum Laden des Views eines Viewcontrollers

Der Zugriff auf die Property in der zweiten Zeile erzwingt das Laden des Views. Den Ladezustand des Views können Sie über die Methode `isViewLoaded` erfragen.

Die Methode `loadView` lädt den View aus der NIB-Datei. Sie können diese überschreiben, um den View durch Code zu erzeugen. Dazu weisen Sie der Property `view` einfach Ihren erstellten View zu. In diesem Fall sollten Sie indes nicht die Methode in der Oberklasse aufrufen. Cocoa Touch ruft nach dem Laden des Views die Methode `viewDidLoad` auf, die Sie ja auch bereits kennengelernt haben.

Tipp

Wenn Sie Ihren Viewcontroller oder dessen View nach dem Laden initialisieren möchten, sollten Sie dazu die Methode `viewDidLoad` überschreiben. Wegen der fehlenden Outlet-Verbindungen ist ja die Methode `awakeFromNib` in den meisten Fällen nicht dafür geeignet. Die Methode `loadView` ist zum Laden des Views gedacht und sollte auch nur dafür überschrieben werden.

Sie können jetzt Ihren Viewcontroller beliebig oft anzeigen und wieder verschwinden lassen. In diesen *Anzeigezyklen* ruft Cocoa Touch `viewWillAppear:`, `viewDidAppear:`, `viewWillDisappear:` und `viewDidDisappear:` auf, die Sie ja bereits kennengelernt haben. Das nächste Kapitel stellt mehrere Möglichkeiten vor, Viewcontroller anzuzeigen und wieder verschwinden zu lassen.

3.2.11 Speicher- und Ressourcenverwaltung des Viewcontrollers

Ein Viewcontroller verwaltet in der Regel mehrere *Ressourcen*. Das sind beispielsweise Views und andere Objekte, die er für seine Aufgabe benötigt und die seinen Zustand abbilden. Dabei besteht ein Unterschied zwischen *wiederherstellbaren* und *nicht wiederherstellbaren Ressourcen*. Zum Beispiel können Sie Views in der Regel wiederherstellen, indem Sie sie aus einem NIB oder Storyboard laden beziehungsweise die Programmanweisungen mit deren Erzeugungscode wieder ausführen. Nicht wiederherstellbare Daten können Nutzereingaben sein, die Sie nicht anderweitig gespeichert haben.

Der Controller kann und sollte wiederherstellbare Ressourcen freigeben, sobald er sie nicht mehr benötigt. Dadurch vermindern Sie den Speicherverbrauch und erhöhen

die Stabilität der App. Der Freigabezeitpunkt hängt dabei in der Regel vom Erzeugungszeitpunkt ab, und die Lebenszeit einer Ressource lässt sich für gewöhnlich einem der drei Zyklen des Viewcontrollers zuordnen:

1. Ressourcen, die der Controller in den Initialisierermethoden oder in `awakeFromNib` erzeugt, sollte er auch erst bei seiner Zerstörung wieder freigeben. Diese Ressourcen haben also den gleichen Lebenszyklus wie der Vewcontroller. Dazu gehören beispielsweise alle Outlets, die in dem NIB liegen, das das Viewcontroller-Objekt enthält. Das sind in Abbildung 3.35 alle Outlets auf der linken Seite.
2. Erstellt der Controller die Ressource beim Laden des Views, so sollten Sie sie auch erst mit dem View freigegeben. Zu den Ressourcen des Ladezyklus gehören beispielsweise der View und die wiederherstellbaren Ressourcen, die Sie in der Methode `viewDidLoad` erzeugen. Die Freigabe erfolgt entweder automatisch oder in einer der Methoden `viewWillUnload`, `viewDidUnload` oder `didReceiveMemoryWarning`. Alle Outlets auf der rechten Seite von Abbildung 3.35 gehören hierzu.
3. Sie können Ressourcen auch innerhalb des Viewzyklus verwalten; also beispielsweise in `viewWillAppear:` anfordern und in `viewWillDisappear:` wieder freigeben.

Diese drei Fälle sind Faustregeln, von denen es aber auch durchaus Ausnahmen geben kann. Wie dem auch sei – Sie sollten die Ressourcen möglichst immer über Propertys verwalten, wie es die Beispielprojekte dieses Buches auch machen. Sie können die Ressourcen immer freigeben, indem Sie die entsprechende Property auf `nil` setzen. Falls Sie automatisches Referenzenzählen einsetzen, sollten Sie für alle Outlets auf Nicht-Top-Level-Objekte – außer für die Arrays von Outlet-Collections – Propertys mit dem Speicherverwaltungstyp `weak` verwenden, da der Controller sie dadurch automatisch mit dem Top-Level-Objekt freigibt.

Top-Level-Objekte und die Arrays von Outlet-Collections müssen Sie hingegen immer halten, indem Sie den Speicherverwaltungstyp `strong` beziehungsweise `retain` verwenden. Außerdem müssen Sie für diese Objekte entscheiden, wann der Viewcontroller sie freigeben soll. Hierbei sind insbesondere die Ressourcen des Ladezyklus interessant. Vor der iOS-Version 6 hat Apple empfohlen, sie in der Methode `viewDidUnload` freizugeben. Diese Methode ist seit iOS 6 indes veraltet, und Sie sollten sie stattdessen in der Methode `didReceiveMemoryWarning` freigeben, wenn die Methode der Oberklasse den View des Controllers freigegeben hat. Der Code sollte also so aussehen:

```
- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    if(![self isViewLoaded]) {
        // Freigabe aller wiederherstellbaren Ressourcen, die
```

```
        // der Controller in viewDidLoad erzeugt oder lädt.
    }
}
```

Listing 3.41 Ressourcenfreigabe unter iOS 6

Da es die Methode `didReceiveMemoryWarning` jedoch schon seit iOS 2 gibt, können Sie das Vorgehen aus Listing 3.41 auch schon für die älteren Betriebssystemversionen einsetzen. Dabei ist die Überprüfung allerdings wichtig. Wenn die App den View des Viewcontrollers während einer Speicherwarnung anzeigt, erhält der Controller zwar die Warnung, aber gibt dennoch nicht seinen View frei.

Tipp

Die Methode `didReceiveMemoryWarning` sollten Sie immer so implementieren, dass Cocoa Touch sie mehrmals aufrufen kann. Sie können die korrekte Funktionsweise übrigens gut im Simulator testen. Rufen Sie dazu einfach den Menüpunkt HARDWARE • SPEICHERWARNHINWEIS SIMULIEREN auf, dann sendet der Simulator eine Speicherwarnung an Ihre App.

3.3 Lokale Benachrichtigungen

Dem Wecker fehlt jetzt nur noch ein Alarmton, der zur gewählten Zeit erklingt. Sie können natürlich die Zeitdifferenz zum aktuellen Zeitpunkt berechnen und einen Timer starten. Dieser ruft nach Ablauf der Zeit eine Methode auf, die einen Ton abspielt. Das funktioniert allerdings nur, wenn Ihr Programm die ganze Zeit im Vordergrund weiterläuft. Sie können also Ihr iPhone in der Zwischenzeit nur eingeschränkt nutzen.

Das Beispielprogramm verwendet stattdessen lokale Benachrichtigungen, die Apple mit iOS 4.0 eingeführt hat. Das sind zeitgesteuerte Ereignisse, die das Betriebssystem zu einem bestimmten Zeitpunkt auslöst. Eine lokale Benachrichtigung kann beim Auslösen eine Nachricht auf dem Bildschirm anzeigen, eine Audiodatei abspielen oder den Wert im *Badge* der App auf einen festen Wert setzen. Die Badges sind die roten Plaketten am App-Icon, die viele Apps (z. B. *Mail*) verwenden (siehe Abbildung 3.36). Lokale Benachrichtigungen funktionieren auch, wenn Sie Ihre App in den Hintergrund schicken oder sogar stoppen.

Vor iOS 5 hat Cocoa Touch die Nachrichten in einer *Alertbox* angezeigt (siehe Abbildung 3.37), und seit iOS 5 erfolgt die Anzeige über die Mitteilungszentrale. Sie stellt lokale Benachrichtigungen standardmäßig als Banner am oberen Bildschirmrand

dar. Sie können in der App *Einstellungen* unter MITTEILUNGEN • ALARMCLOCK[7] • HINWEISSTIL aber HINWEISE auswählen, um die Nachrichten in Alertboxen anzeigen zu lassen. Eine (Vor-)Auswahl dieses Anzeigestils über die App ist dagegen leider nicht möglich.

Abbildung 3.36 App-Icons mit Badges

Nachrichten an aktive Apps

Die Nachricht erscheint dagegen nicht, wenn die App im Vordergrund läuft. Stattdessen ruft Cocoa Touch eine Methode des Applicationd-Delegates auf. In diesem Fall können Sie also selbst entscheiden, was Sie mit der Nachricht anfangen wollen.

Abbildung 3.37 Anzeige einer lokalen Benachrichtigung in einer Alertbox ...

Abbildung 3.38 ... und in einem Banner

Sie definieren eine neue Benachrichtigung, indem Sie ein Objekt der Klasse `UILocalNotification` erzeugen und den Auslösezeitpunkt setzen. Außerdem können Sie

7 Nach der Lokalisierung der App in Abschnitt Sprachkursus für die App heißt dieser Punkt Wecker.

einen Nachrichtentitel und -text angeben, wenn Sie eine Alertbox anzeigen lassen wollen. Für den Benachrichtigungston und das Badge können Sie den Namen des Tons beziehungsweise den Wert übergeben. Die neue Benachrichtigung fügen Sie danach in die *Warteschlange* für lokale Benachrichtigungen ein, auf die Sie über Methoden des Application-Singletons zugreifen können. Cocoa Touch liest die Benachrichtigungen von dort, löst sie zum gewünschten Zeitpunkt aus und löscht sie danach, falls es sich um eine einmalige Benachrichtigung handelt.

3.3.1 Benachrichtigungen versenden

Die Methode `scheduleLocalNotification:` der Klasse `UIApplication` fügt eine Benachrichtigung in die Warteschlange ein, und `cancelLocalNotification:` entfernt sie wieder daraus; die Methode `cancelAllLocalNotifications` entfernt alle Benachrichtigungen der App aus der Warteschlange. Über die Property `scheduledLocalNotifications` können Sie alle Benachrichtigungen aus der Warteschlange auslesen, und ab iOS 4.2 können Sie sie auch setzen. Sie können in der Warteschlange allerdings nur auf die Benachrichtigungen Ihrer App zugreifen. Der Zugriff auf die Benachrichtigungen anderer Apps ist nicht möglich. Ebenso wenig können Sie anderen Apps über diesen Mechanismus eine Benachrichtigung schicken.

Die Methoden aus Abschnitt 3.2.8, »Direkte Gestenverarbeitung«, haben die Methode `updateAlarm` aufgerufen, um das Auslösen des Alarms zu aktualisieren. Diese Methode löscht entweder alle Benachrichtigungen der App, wenn Sie den Alarm ausgeschaltet haben, oder legt über einen Aufruf der Methode `createAlarm` eine neue Benachrichtigung an.

```
- (IBAction)updateAlarm {
    self.alarmHidden = !self.alarmSwitch.on;
    if(self.alarmSwitch.on) {
        [self createAlarm];
    }
    else {
        UIApplication *theApplication =
            [UIApplication sharedApplication];

        [theApplication cancelAllLocalNotifications];
    }
}
```

Listing 3.42 Aktualisierung des Alarms

Die Methode `createAlarm` in der Klasse `AlarmClockViewController` erzeugt eine neue Benachrichtigung:

```
const NSTimeInterval kSecondsOfDay = 60.0 * 60.0 * 24.0;

- (void)createAlarm {
    UIApplication *theApplication =
        [UIApplication sharedApplication];
    UILocalNotification *theNotification =
        [[UILocalNotification alloc] init];
    NSTimeInterval theTime = self.startTimeOfCurrentDay +↩
                             clockControl.time;
    while(theTime < [NSDate timeIntervalSinceReferenceDate]) {
        theTime += kSecondsOfDay / 2.0;
    }
    [theApplication cancelAllLocalNotifications];
    theNotification.fireDate =
         [NSDate dateWithTimeIntervalSinceReferenceDate:theTime];
    theNotification.timeZone = [NSTimeZone defaultTimeZone];
    theNotification.alertBody = @"Aufwachen";
    theNotification.soundName = ↩
        UILocalNotificationDefaultSoundName;
    [theApplication scheduleLocalNotification:theNotification];
    [theNotification release];
}
```

Listing 3.43 Erzeugung einer lokalen Benachrichtigung

Zuerst muss die Methode die Alarmzeit in ein `NSDate`-Objekt umwandeln. Die Alarmzeit liegt als relative Zeit zum Beginn des aktuellen Tages oder der letzten Mittagszeit vor, da das Ziffernblatt nur 12 und nicht 24 Stunden umfasst. Die Berechnung erfolgt nun so, dass zu der Startzeit des Tages die Alarmzeit hinzugerechnet wird. Solange diese Zeit noch kleiner als die aktuelle Zeit ist, zählt die Methode jeweils die Sekunden eines halben Tages hinzu. Dabei enthält die Konstante `kSecondsOfDay` die Anzahl der Sekunden eines Tages als Zeitintervall. Bevor Sie jedoch die neue Alarmzeit setzen, müssen Sie die eventuell noch vorhandenen Benachrichtigungen aus der Warteschlange löschen. Als Alarmton verwenden Sie den Systemton für Benachrichtigungen, indem Sie der Property `soundName` die Konstante `UILocalNotificationDefaultSoundName` zuweisen.

Die Methode `startTimeOfCurrentDay` berechnet den Startzeitpunkt des Tages zu der aktuellen Uhrzeit. Dazu verwenden Sie einen Kalender, mit dem Sie das aktuelle Datum ohne Uhrzeit in einem `NSDateComponents`-Objekt berechnen lassen. Da alle Zeitkomponenten auf 0 stehen, hat dieses Objekt die Uhrzeit 0:00 Uhr. Sie brauchen es also nur über den Kalender wieder in ein `NSDate`-Objekt zurückrechnen lassen:

```
- (NSTimeInterval)startTimeOfCurrentDay {
    NSCalendar *theCalendar = [NSCalendar currentCalendar];
    NSDateComponents *theComponents = ➲
        [theCalendar components:NSYearCalendarUnit |➲
                                NSMonthCalendarUnit |➲
                                NSDayCalendarUnit➲
                        fromDate:[NSDate date]];
    NSDate *theDate =➲
        [theCalendar dateFromComponents:theComponents];

    return [theDate timeIntervalSinceReferenceDate];
}
```

Listing 3.44 Berechnung des Startzeitpunktes des aktuellen Tages

Damit haben Sie den Versand einer lokalen Benachrichtigung abgeschlossen. Denken Sie beim Ausprobieren der App aber daran, dass sie die Benachrichtigungen nicht anzeigt, wenn die App im Vordergrund läuft. Sie sollten die App also vor der Versendung über den Homebutton des Simulators in den Hintergrund schicken oder sie über Xcode stoppen.

Die App soll nach einem Neustart natürlich auch wieder die aktuelle Alarmzeit anzeigen. Dazu muss sie vor der Anzeige des Views die letzte Benachrichtigung auslesen und deren Zeit auf das Clockcontrol und das Label für die Anzeige der Alarmzeit übertragen, was die Methode `updateViews` übernimmt.

```
- (void)updateViews {
    UIApplication *theApplication = ➲
        [UIApplication sharedApplication];
    UILocalNotification *theNotification = [[theApplication➲
        scheduledLocalNotifications] lastObject];

    if([theNotification.fireDate compare:[NSDate date]] >➲
        NSOrderedSame) {
        NSTimeInterval theTime = [theNotification.fireDate➲
            timeIntervalSinceReferenceDate] - ➲
                self.startTimeOfCurrentDay;

        self.clockControl.time = ➲
            remainder(theTime, kSecondsOfDay / 2.0);
        self.alarmHidden = NO;
    }
    else {
        self.alarmHidden = YES;
```

```
    }
    [self updateTimeLabel];
}
```

Listing 3.45 Aktualisierung der Views aus den Benachrichtigungen

Sie berechnet aus der absoluten Alarmzeit und der Startzeit des aktuellen Tages die relative Alarmzeit zum aktuellen Tag und übergibt diesen Wert an das Clockcontrol. Außerdem sorgt sie durch einen Aufruf von `updateTimeLabel` dafür, dass der Controller auch die Textanzeige aktualisiert. Um diese Aktualisierung zu aktivieren, müssen Sie die Methode noch aus der Methode `viewWillAppear:` aus aufrufen:

```
- (void)viewWillAppear:(BOOL)inAnimated {
    [super viewWillAppear:inAnimated];
    [self updateViews];
}
```

Listing 3.46 Aufruf der Aktualisierungsmethode

3.3.2 Benachrichtigungen verarbeiten

Durch eine lokale Benachrichtigung können Sie folgende Ereignisse auslösen. Sie können

- eine Nachricht anzeigen lassen,
- einen Klang mit einer maximalen Länge von 30 Sekunden abspielen und
- die Zahl in dem Badge, das am Icon der Applikation im Springboard erscheint, auf einen festen Wert setzen.

Die lokale Benachrichtigung kann indes nicht Ihre Applikation automatisch in den Vordergrund bringen. Entweder ist sie bereits im Vordergrund, oder abhängig von der Konfiguration erscheinen eine Nachricht und die Badgenummer und ertönt der Klang. Um die App zu aktivieren, muss der Nutzer den Button ANZEIGEN in der Alertbox betätigen beziehungsweise das Banner antippen.

Wenn die App im Vordergrund läuft, informiert Cocoa Touch Sie direkt über das Auftreten einer lokalen Benachrichtigung. Das geschieht über die Methode `application:didReceiveLocalNotification:` im Application-Delegate. Das Betriebssystem ruft diese Methode auch auf, wenn die App nicht oder im Hintergrund lief und der Nutzer die App über die Nachricht aktiviert hat. Sie können diese Methode nutzen, um auch dann eine Nachricht anzuzeigen, wenn die App während des Nachrichtenempfangs aktiv ist. Mit dieser Methode können Sie also die Benachrichtigung manuell verarbeiten. Das Beispielprogramm soll eine Alertbox anzeigen und einen Ton abspielen.

Allerdings sollten Sie die Alertbox auch nur dann anzeigen, wenn die App bereits aktiv war. Andernfalls sähe der Nutzer ja zwei Hinweise auf den Alarm direkt hintereinander, was er zumindest als störend oder sogar fehlerhaft ansehen könnte. Um diesen Effekt zu vermeiden, können Sie den Zustand der App beim Auslösen der Benachrichtigung über die Property `applicationState` des Singletons `UIApplication` unterscheiden. Nur wenn die App den Zustand `UIApplicationStateActive` hat, soll sie den Alarm auslösen.

Die Nachricht zeigen Sie über eine Alertbox an. Dazu verwenden Sie die Klasse `UIAlertView`. Die Alertbox in Listing 3.47 zeigt nur die Nachricht und einen OK-Button an und lässt den Titel leer.

```
- (void)application:(UIApplication *)inApplication➲
    didReceiveLocalNotification:➲
        (UILocalNotification *)inNotification {
    if(inApplication.applicationState ==➲
        UIApplicationStateActive) {
        UIAlertView *theAlert =➲
            [[UIAlertView alloc] initWithTitle:nil➲
                message:inNotification.alertBody➲
                delegate:nil cancelButtonTitle:@"OK"➲
                otherButtonTitles:nil];
        [theAlert show];
        [theAlert release];
        [self playSound];
    }
}
```

Listing 3.47 Anzeige einer Alertbox beim Empfang einer Benachrichtigung

Sie können den Systemton für lokale Benachrichtigungen leider nicht selbst in Ihrem Programm abspielen. Es gibt dafür keine Funktion oder Methode, die genau diesen Klang ertönen lässt. Die Applikation soll stattdessen einen anderen Ton abspielen, was in der Methode `playSound` geschieht. Das Klingeln eines mechanischen Weckers passt hier sehr gut, und Sie finden die Datei *ringtone.caf* im Beispielprojekt auf der DVD. Sie können aber auch andere Töne verwenden, wenn sie im passenden Format vorliegen.

Die Töne müssen laut Dokumentation als 16-Bit-Little-Endian-PCM-Audiodaten im *Core-Audio-Format* mit der Endung *.caf* vorliegen und die Länge des Tons darf nicht größer als 30 Sekunden sein. Unter OS X gibt es das Kommandozeilenprogramm `afconvert`, mit dem Sie eine Audiodatei in dieses Format konvertieren können. Der Aufruf dafür ist:

```
/usr/bin/afconvert -f caff -d LEI16 {Eingabedatei} {Ausgabedatei}
```

Dabei ist der Parameter {Eingabedatei} die Datei mit dem Ton im Ursprungsformat, und das Programm schreibt den konvertierten Klang in die Datei {Ausgabedatei}. Liegt der Klingelton für den analogen Wecker beispielsweise als WAV-Datei vor, können Sie ihn mit

```
/usr/bin/afconvert -f caff -d LEI16 ringtone.wav ringtone.caf
```

in eine CAF-Datei konvertieren.

Bevor Sie den Klang verwenden können, müssen Sie diesen als Ressource zu Ihrem Projekt hinzufügen. Dazu ziehen Sie einfach die gewünschte Datei auf die Gruppe Supporting Files in der Navigatorspalte des Projekts. Sie sollten im folgenden Dialog darauf achten, dass Xcode die Datei auch in das Projekt kopiert und dem Target AlarmClock zuordnet. Setzen Sie dazu die Häckchen in den Checkboxen Destination und Add to targets. Sie können anschließend den Systemton bei der Erzeugung der lokalen Benachrichtigung auch durch die Sounddatei ersetzen, indem Sie den Dateinamen ohne Pfadangabe verwenden. Tauschen Sie dazu einfach die Zuweisung des Klangnamens in Listing 3.43 durch die folgende Anweisung aus:

```
theNotification.soundName = @"ringtone.caf";
```

Im Gegensatz zu dem Systemton können Sie diesen Klang auch über Funktionen aus der Audio-Toolbox abspielen., wozu Sie allerdings dieses Framework einbinden müssen. Wählen Sie dazu zunächst das Target der App aus. Ein Xcode-Projekt kann mehrere Produkte (zum Beispiel Apps oder Frameworks) verwalten. Jedes Produkt hat dabei sein eigenes Target. Es legt die Dateien und Regeln zur Erstellung des jeweiligen Produkts fest. Sie erreichen das Target, wenn Sie in der linken Navigatorspalte das Projekt auswählen. Daneben erscheint eine Spalte mit dem Abschnitt Targets. Wählen Sie dort das Target AlarmClock aus (siehe Abbildung 3.39, links).

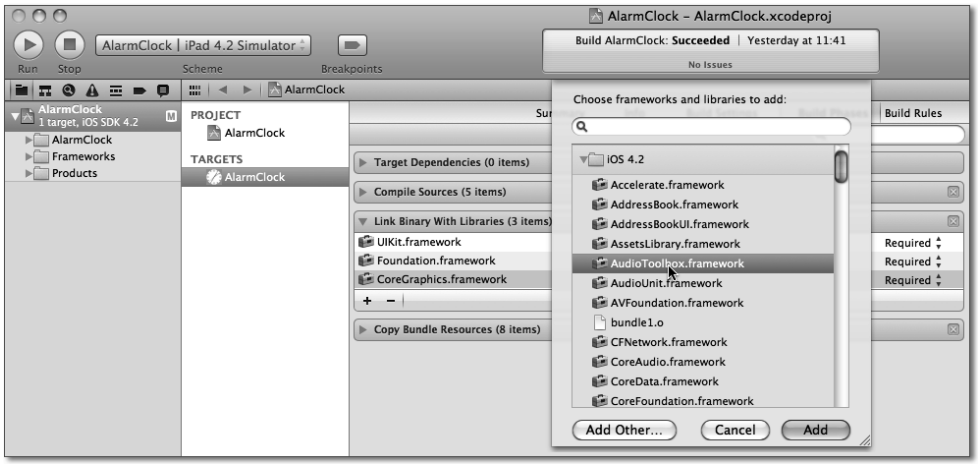

Abbildung 3.39 So fügen Sie ein Framework zu einem Target hinzu.

Im Hauptfenster befindet sich dann ein Tabulator BUILD PHASES mit dem Abschnitt LINK BINARY WITH LIBRARIES. Öffnen Sie diesen Punkt durch einen Klick auf das entsprechende Dreieck. Sie können dem Target nun über das Plus-Zeichen neue Frameworks hinzufügen. Selektieren Sie im Dialog den Eintrag AUDIOTOOLBOX.FRAMEWORK, und klicken Sie auf den Button ADD.

Wenn Sie ein Framework einbinden, fügt Xcode sowohl die Binärdateien mit den entsprechenden Klassen, Methoden und Funktionen als auch die entsprechenden Suchpfade für Headerdateien mit den Deklarationen zum Projekt hinzu. Dabei liegen die Headerdateien immer unterhalb eines virtuellen Ordners mit dem Namen des Frameworks.

In der Regel hat jedes Framework von Apple eine Headerdatei mit dem Namen des Frameworks. Über diese Datei können Sie alle Header des Frameworks auf einmal einbinden. Da Sie die Audio-Toolbox im Application-Delegate verwenden möchten, müssen Sie dort den Header dieses Frameworks über die Direktive

```
#import <AudioToolbox/AudioToolbox.h>
```

einbinden.

Das Abspielen des Tons erfolgt in zwei Schritten. Zuerst muss ihn die App aus der Datei laden. Sie erhalten dabei einen Verweis auf den Ton, mit dem Sie ihn beliebig oft abspielen können. Wenn Sie ihn nicht mehr benötigen, sollten Sie den Ton über diesen Verweis auch wieder freigeben.

Der Verweis auf den Ton ist eine vorzeichenlose Zahl, die das Application-Delegate über die Property `soundId` verwaltet. Das Delegate soll den Ton jedoch nur einmal laden, und das nur dann, wenn es ihn braucht. Dies erfolgt in dem *Lazy-Getter* `soundId`. Diese Methode prüft zuerst, ob sie den Ton bereits geladen hat, und macht dies gegebenenfalls.

Lazy-Getter

Lazy-Getter sind eine einfache und effektive Möglichkeit für den sparsamen Umgang mit Ressourcen. Sie haben bereits ein weiteres Beispiel dafür kennengelernt. Die Property `view` der Klasse `UIViewController` lädt den View erst, wenn das Programm das erste Mal darauf zugreift. Das ist in der Regel kurz vor dessen Anzeige.

Lazy-Getter haben meist den gleichen, einfachen Aufbau:

```
- (MyClass *)myObject {
    if(myObject == nil) {
        myObject = ...; // Speicherverwaltung nicht vergessen!
    }
    return myObject;
}
```

Die Methode `playSound` spielt den Ton ab:

```
- (NSNumber *)soundId {
    if(soundId == nil) {
        NSURL *theURL =
           [[NSBundle mainBundle] URLForResource:@"ringtone"
                                   withExtension:@"caf"];
        SystemSoundID theId;

        if(AudioServicesCreateSystemSoundID(
            (CFURLRef) theURL, &theId) ==
                kAudioServicesNoError) {
            self.soundId =
                [NSNumber numberWithUnsignedInt:theId];
        }
    }
    return soundId;
}
- (void)playSound {
    NSNumber *theId = self.soundId;

    if(theId) {
        AudioServicesPlaySystemSound(
            [theId unsignedIntValue]);
    }
}
```

Listing 3.48 Laden und Abspielen eines Tons

Listing 3.48 verwendet für die Property `soundId` anstatt des Typs `NSUInteger` die Klasse `NSNumber`. Das hat den Vorteil, dass Sie über den Property-Wert `nil` den Zustand des nicht geladenen Tons abbilden können. In der Methode `dealloc` müssen Sie den Ton auch wieder freigeben, damit kein Speicherleck entsteht. Das können Sie durch den folgenden Setter erreichen:

```
- (void)setSoundId:(NSNumber *)inSoundId {
    if(soundId != inSoundId) {
        if(soundId != nil) {
            AudioServicesDisposeSystemSoundID(
                [soundId unsignedIntValue]);
            [soundId release];
        }
```

```
        soundId = [inSoundId retain];
    }
}
```

Listing 3.49 Freigabe des Alarmtons im Setter

Hier erfolgt der lesende Zugriff auf den Wert auch über das Attribut und nicht über den Getter. Der wäre hier kontraproduktiv, da er ja den Ton lädt, falls die App ihn noch nicht geladen hat. Diese Implementierung stellt außerdem sicher, dass das Delegate den alten Alarmton immer freigibt, wenn Sie einen neuen über den Setter setzen.

Das Application-Delegate kann über die Methode `applicationDidReceiveMemoryWarning:` auch auf Speicherwarnungen reagieren. Da es den Alarmton jederzeit wieder laden kann, sollten Sie in dieser Methode den Property-Wert für `soundId` auf `nil` setzen, um den Ton freizugegeben und dadurch Speicherplatz einzusparen.

```
- (void)applicationDidReceiveMemoryWarning:↵
    (UIApplication *)inApplication {
    self.soundId = nil;
}
```

Listing 3.50 Freigabe des Alarmtons bei einer Speicherwarnung

Ressourcen sind vielfältig

Auch der Alarmton stellt eine Ressource des Programms dar, die nach der Terminologie aus Abschnitt 3.2.11 wiederherstellbar ist. Dabei besteht zwar die Referenz auf den Ton nur aus einigen Bytes, die Tondaten dürften aber wesentlich größer sein.

3.4 Eine App für alle

Im *Springboard*, das ist der Home-Screen unter iOS, über den Sie Apps starten, erscheint für Ihre App nur ein weißes, abgerundetes Quadrat mit einem Glanzeffekt und dem Namen *AlarmClock*. Sie können natürlich auch ein eigenes Icon anzeigen lassen. Das Bild muss im PNG-Format in den folgenden Größen vorliegen:

Display-Typ	Format	Bemerkung
Standard	57 × 57 Pixel	Notwendig für iPhone und iPod touch
iPhone-Retina	114 × 114 Pixel	Optional für iPhone und iPod touch mit Retina-Display

Tabelle 3.3 Formate für das App-Icon

Index

!= 180
#import 92
#include 92
#pragma clang diagnostic 806
% 104, 215
& 137, 178
* 137
== 180
@ 177, 183
@( ) 184
@[ ] 184
@{ } 186
@2x 281
@autoreleasepool 150, 168
@avg 160
@class 119
@count 160
@dynamic 398
@end 66
@implementation 67
@interface 66
@max 160
@min 160
@optional 117, 132
@property 89
@protocol 117
Vorwärtsdeklaration 119
@required 117
@sum 160
@synthesize 89
__bridge 597, 638, 782
__bridge_retained 597, 638
__bridge_transfer 597, 638, 782
__strong 164
__unsafe_unretained 167
__weak 165
_cmd 82

A

Abhängigkeit 74
Ableiten 61
Abstrakte Klasse 64
Abstraktion 58
accelerometer:didAccelerate: 539
AccelerometerGraph 539
Accept-Ranges 840
Accessor 60, 88, 647
Achsangaben 817
Action 227
actionForKey: 603
actionForLayer:forKey: 603, 605
Actionmethode 98, 107, 236
actions 603
Activityindicator 463
Adams, Douglas 690
addAnimation:forKey: 591
addAnnotation: 876
addAnnotations: 876
addChildViewController: 387
addConstraint: 425
addConstraints: 425
addGestureRecognizer: 255
addObject: 97
addObserver:forKeyPath:options:
context: 120, 543
addSublayer: 571
addSubview: 612
addTarget:action: 256
Ad-Hoc-Distribution 913, 946
adjustsImageWhenHighlighted 242
afconvert (Kommandozeilenprogramm) 274
Affine Transformation 560
Agil 975
Aktion 941
Aktive Applikation 129
Alertbox 268
Alertview 193
alertViewStyle 846
allCredentials 849
allHeaderFields 830
allObjects 188
alloc 70, 83, 85, 136
Allocations 642
allTouches 240, 248
alpha 560, 572
Analyzer 162, 292
Anfrage
asynchrone 820
synchrone 820
Angemessenheit 982
Angriff 661
animatesDrop 877
animateWithDuration:animations: 558

animateWithDuration:animations:
completion: 559
animateWithDuration:delay:options:
animations:completion: 563
animation 592
animations 595
Animierbare Property 560
Anmerkungen auf einer Karte 875
annotations 876
Anonyme Kategorie 116, 155, 222
ANSI-C 65
Antialiasing 622
Antipattern 169
anyObject 188
Anzeigeausrichtung 416
App lokalisieren 282
App Store 22
appearance 400
appearanceWhenContainedIn: 400
appendFormat: 180
appendString: 181
App-Icon 278
App-ID 29, 51
Apple Developer Network 22
Apple Push Notification 44
application:didFinishLaunching-
WithOptions: 128, 261, 290, 312, 317
application:didReceiveLocal-
Notification: 273, 408, 942
application:supportedInterface-
OrientationsForWindow: 368
applicationBecomeActive: 129
Application-Delegate 126, 225, 379
applicationDidEnterBackground: 129
applicationDidReceiveMemory-
Warning: 130, 131, 278
Application-Singleton 322
applicationState 274
applicationWillTerminate: 129
Äquator 871
ARC 32, 135, 162
Archiv 913
Archivverwaltung 913
ARMv6 916
ARMv7 916
ARP 149
Array und Zahlen 183
arrayWithContentsOfFile: 793
arrayWithContentsOfURL: 793, 822
arrayWithObjects: 186
AS/NZS 665
assign 154
Assistant Editor 889
atan2 250
Atomar 177
Attack Surface Reduction 697
Attribut 59, 66
indiziertes 437
optionales 437
transientes 436
Attributinspektor 36, 98, 210
Audio-Toolbox 275
Aufrufstapel 644
Ausdruck JSON 771
Ausnahme 304
Aussagekräftiger Name 196
Authentisierung 690
Autolayout 419
ausschalten 411
Autolayout-Constraint 419
Automatic Reference Counting 32, 135, 162
automaticallyForwardAppearance-
AndRotationMethodsToChild-
ViewControllers 390
Automatisches Referenzen-
zählen 149, 155, 162
für einzelne Dateien ausschalten 784
und Blockfunktionen 192
und Speicherlecks 642
und Toll-Free Bridge 637
und Zombies 650
autorelease 142
Autorelease-Pool 142, 149, 467, 782
Autoresizingmask 410, 411
und Restriktionen 427
autoreverses 591
Autorisierung 690
Autosizing 203, 290, 410, 411
Autovervollständigung 890
AVAudioPlayer 462
AVAudioRecorder 469
AVAudioSession 469
awakeFromNib 220, 262, 267, 369, 480, 580
Awareness 977

B

Background 129
backgroundColor 560
backgroundImage 381
Backup 979
Badge 268, 373

badgeValue ... 374
Base Internationalization ... 431
Basic Authentication ... 822
Basiskonfiguration ... 932
Baumdarstellung ... 234
Bedrohung ... 661
Bedrohungsanalyse ... 664
Bedrohungsdatenbank ... 984
beginAnimations:context: ... 565
beginAppearanceTransition:animated: ... 390
beginRefreshing ... 779
beginTime ... 600
beginTrackingWithTouch:withEvent: ... 248
Beispielprojekt
AlarmClock ... 199, 316, 430
AlarmClock mit Storyboard ... 316, 347
Animation ... 564
AutoLayout ... 428
Bars ... 345
Container ... 386
FlipBookAnimation ... 596
Fototagebuch ... 217
Games ... 372, 374, 529, 906
GetURL ... 32
Instruments ... 641
Layer ... 577
Modal ... 326
More ... 376, 377
Page ... 383
PDFView ... 630
PhotoDiary ... 405
Pie ... 571, 609
RefactoredAlarmClock ... 906
Retina ... 281
Rotation ... 414
ScrollView ... 613
SiteSchedule ... 798, 836
Subview ... 392
TableView ... 474
UniversalYouTube ... 507
Unwind ... 342
XPath ... 815
YouTube ... 772
Benachrichtigung ... 130, 406, 489
lokale ... 268
Benutzercode ... 718
Beobachter ... 120
Bereichsanfrage ... 840
Bereitstellungsprofil ... 43
Berners-Lee, Tim ... 65
Berührungsgeste ... 239
Beschleunigungssensor ... 538
Betriebsdokumentation ... 981
Betriebsumgebung ... 988
Bewegungssensor ... 25
Bezeichner ... 196
Bibliothek ... 922
einbinden ... 930
Bibliotheken ... 985
Bildgröße
logische ... 280
physikalische ... 280
Binäranalyse ... 986
Bitmaske ... 242, 410
Bitmenge ... 563
Bitweises Oder ... 369, 563
Blacklist ... 985
Blätterachse ... 381
Block ... 189
Blockfunktion ... 189
Bogenmaß ... 212
Bouncen ... 775
Boundary ... 856
Bounds ... 207
Boxing ... 183
Branch ... 958
Breakpoint ... 294, 885
Breakpoint Navigator ... 295, 887
Breakpointleiste ... 890
Buffer Overflow ... 663
Bugtracking ... 978
Buildlog ... 887
Build-Settings ... 915
Buildsystem ... 919
Bundle Identifier ... 51, 914
Bundle Seed ID ... 51
Button ... 36
bytes ... 184
Bytesequenz ... 766
byValue ... 592

C

c't ... 655
CAAction ... 603
CAAnimation ... 591
CAAnimationGroup ... 595
CABasicAnimation ... 592
CACurrentMediaTime ... 600
CAGradientLayer ... 577
CAKeyframeAnimation ... 592, 593
CALayer ... 570

calculationMode ... 598
Callback ... 194, 868
CAMediaTiming ... 598
canCancelContentTouches ... 618
cancel ... 836
cancelAllLocalNotifications ... 270
cancelAuthenticationChallenge: ... 846
cancelLocalNotification: ... 270
cancelTrackingWithEvent: ... 248
canPerformUnwindSegueAction:from-ViewController:withSender: ... 341
canShowCallout ... 877
CAPropertyAnimation ... 592
CAReplicationLayer ... 589
Carriage-Return Line-Feed ... 858
CAScrollLayer ... 579
CAShapeLayer ... 579
CATextLayer ... 578
CATiledLayer ... 623
CATransform3D ... 590, 592, 608
CATransform3DMakeRotation ... 609
CATransition ... 594
CDATA-Abschnitt ... 789
center ... 560
centerCoordinate ... 871
Certificate Signing Request ... 44
Certificates ... 719
CFBridgingRelease ... 639
CFBridgingRetain ... 639
CFRelease ... 635, 638
CFRetain ... 635, 638
CFStringConvertNSStringEncoding-ToEncoding ... 781
CFURLRef ... 597
CGAffineTransform ... 560
CGAffineTransformIdentity ... 561
CGAffineTransformMakeScale ... 609
CGColorRef ... 577
CGContextAddArc ... 215
CGContextAddEllipseInRect ... 209
CGContextClip ... 209, 214
CGContextClipToMask ... 584
CGContextDrawPDFPage ... 630
CGContextFillPath ... 209
CGContextMoveToPoint ... 215
CGContextRef ... 208
CGContextRestoreGState ... 209
CGContextSaveGState ... 209
CGContextSelectFont ... 588
CGContextSetAllowsAntialiasing ... 623
CGContextSetLineCap ... 213
CGContextSetRGBAFillColor ... 209
CGContextSetRGBStrokeColor ... 213
CGContextSetTextMatrix ... 588
CGContextStrokePath ... 209
CGImage ... 572
CGImageRef ... 572, 597
CGImageSourceRef ... 597
CGMutablePathRef ... 580
CGPathRef ... 580
CGPDFDocumentCreateWithURL ... 636
CGPDFDocumentGetNumberOfPages ... 631
CGPDFDocumentGetPage ... 631
CGPDFPageGetDrawingTransform ... 630
CGRect ... 205
CGRectInset ... 632
CGRectIntersectsRect ... 520
CGRectOffset ... 349
CGRectUnion ... 518
CGRectValue ... 449
characterAtIndex: ... 179
charValue ... 69
class ... 69, 73
Clean Desk Policy ... 979
CLGeocoder ... 873
Client ... 765
Clippen ... 209
Clipping ... 213
clipsToBounds ... 247, 572
CLLocationCoordinate2D ... 871
Closure ... 189
CLPlacemark ... 875
Cocoa Touch ... 55
Code Signing ... 43
Code Snippet Library ... 888, 891
Code-Analyse ... 986
Code-Duplikat ... 61
Code-Kommentierung ... 985
Code-Sign-Fehler ... 52
Code-Smell ... 61, 904
Coding Guidelines for Cocoa ... 196
Collection ... 174
Collectionview ... 507
collectionView:cellForItemAtIndexPath: ... 513
collectionView:didSelectItemAt-IndexPath: ... 526
collectionView:layout:sizeForItemAt-IndexPath: ... 511, 522
collectionView:numberOfItemsIn-Section: ... 513
collectionView:viewForSupplementary-ElementOfKind:atIndexPath: ... 514

collectionViewContentSize ... 518
columnNumber ... 809
Commit ... 957
commitAnimations ... 565
Company iOS Developer Programm ... 23
Completionblock ... 569
Compliance ... 982
components:fromDate: ... 218
componentsJoinedByString: ... 185
Concurrent Versions System ... 75
connection:didFailWithError: ... 835
connection:didReceiveAuthentication-
Challenge: ... 846
connection:didReceiveData: ... 835
connection:didReceiveResponse: ... 835
connection:didSendBodyData:totalBytes-
Written:totalBytesExpectedToWrite: ... 835
connection:willSendRequestFor-
AuthenticationChallenge: ... 846
Connection-Delegate ... 834
connectionDidFinishLoading: ... 835
connectionWithRequest:delegate: ... 834
constraintsWithVisualFormat:options:
metrics:views: ... 427
constraintWithItem:attribute:relatedBy:
toItem:attribute:multiplier:constant: ... 426
Container ... 174
Containerview ... 233
Containerviewcontroller ... 333, 386
containsObject: ... 188
Content-Disposition ... 857
contentInset ... 612
Content-Length ... 827
contentOffset ... 612
contents ... 572, 584, 597
contentSize ... 612
contentStretch ... 560
Content-Type ... 827
contentView ... 475
Continue ... 883
continue ... 299
continueTrackingWithTouch:withEvent: ... 248
continueWithoutCredentialFor-
AuthenticationChallenge: ... 846
Control ... 236
Controller ... 73
controller:didChangeObject:atIndexPath:
forChangeType:newIndexPath: ... 491
controller:didChangeSection:atIndex:
forChangeType: ... 493
controllerDidChangeContent: ... 491
Controllerschicht ... 307
controllerWillChangeContent: ... 491
Controlzustand ... 242
Convenience-Konstruktor ... 84, 144
convertPoint:fromView: ... 246
convertPoint:toView: ... 246
convertRect:toView: ... 450
convertTime:fromLayer: ... 600
coordinate ... 875
copy ... 154, 176, 186, 192
copyWithZone: ... 156
Core Animation ... 570
Core Data ... 158, 355, 434, 798
Core Foundation ... 635, 637
Speicherverwaltungsregeln ... 635
Core Graphics ... 208, 635
Core-Graphics-Pfad ... 579
cornerRadius ... 572
count ... 185
CPSSE ... 976
Crash ... 883
Crashreport ... 912, 913
CRLF ... 858
CSR ... 44
curl ... 827
currentCalendar ... 220
Customlayout ... 509
customView ... 338
CVS ... 75
CXMLDocument ... 813
CXMLElement ... 813
CXMLNode ... 813

D

Dahl, Ole-Johan ... 58
Dangling Pointer ... 138, 305, 647
dataUsingEncoding: ... 769
dataWithBytes:length: ... 184
dataWithContentsOfFile:options:error: ... 184
dataWithContentsOfURL: ... 773, 822, 850
dataWithContentsOfURL:options:
error: ... 184, 822
dataWithJSONObject:options:error: ... 783
dataWithPropertyList:format:
options:error: ... 794
date ... 84
Dateiattribut ... 732
Dateiinspektor ... 283
Dateisysteme ... 26
Datei-Upload ... 856

Datenfluss ... 671
Datenflussdiagramm ... 665
Datenkapselung ... 58, 79
Datenmodellinspektor ... 436
Datenquelle ... 382, 384, 472, 488
Datensparsamkeit ... 698
Datenspeicher ... 671
Datumsdarstellung ... 217
Datumsobjekt ... 217
dealloc ... 148
Debug Navigator ... 887
Debug-Ansicht ... 41
Debugger ... 294
Debuggerbefehl ... 298
Debuggerbereich ... 293
Debug-Information ... 987
Debug-Konfiguration ... 930
Debugsymbol ... 913
Decorationview ... 507
defaultActionForKey: ... 603
defaultCenter ... 406
defaultValueForKey: ... 575, 602
Defense in Depth ... 698
Delays Content Touches ... 622
delaysContentTouches ... 622
Delegate ... 126
delegate ... 572
Property ... 126
delegate method ... 572
Delegate-Protokoll ... 132, 396
Delegation ... 74, 125
und Responder-Methoden ... 248
deleteRowsAtIndexPaths:withRow-Animation: ... 492
deleteSections:withRowAnimation: ... 493
Denial of Service ... 690, 696
Deployment ... 987
dequeueReusableCellWithIdentifier: ... 473
dequeueReusableCellWithReuseIdentifier: forIndexPath: ... 513
dequeueReusableSupplementaryView-OfKind:withReuseIdentifier: forIndexPath: ... 515
Dereferenzieren ... 137
Derived Data ... 913
description ... 182, 294, 300, 853
descriptionWithLocale: ... 182
Deseralisierung ... 765
Design ... 983
Designfehler ... 663
Design-Grundsätze ... 697
Designierter Initializer ... 93
Detailviewcontroller ... 362
Dictionary ... 186
und Zahlen ... 183
dictionaryWithContentsOfFile: ... 793
dictionaryWithContentsOfURL: ... 793, 822
dictionaryWithObjectsAndKeys: ... 186
didChangeValueForKey: ... 125
didMoveToParentViewController: ... 387
didReceiveMemoryWarning ... 130, 267
didRotateFromInterfaceOrientation: ... 389, 416
Diff ... 955
Digest Authentication ... 822
digit ... 586
dismissPopoverAnimated: ... 330
display ... 302
Distributionszertifikat ... 915
Divisionsrest ... 104
Divisionsrestoperator ... 215
Document Object Model ... 811
Dokumentation ... 897, 916, 981
DOM ... 795, 811
DOM-Baum ... 812
Doppeltap ... 241
Doppelter Code ... 61
doubleSided ... 381
drain ... 150
drawAsPatternInRect: ... 464
drawInContext: ... 573
drawInRect:withFont: ... 349
drawLayer:inContext: ... 573, 587, 628
drawRect: ... 208, 345, 627
DUNS-Nummer ... 23
duration ... 591
Dynamischer Nachrichtenversand ... 805
Dynamisches Binden ... 65

E

Ebene ... 73
Ebene 1 ... 668
Eigentümer einer NIB-Datei ... 227
Eigentümerschaft ... 638
Elementarer Datentyp ... 95
Elevation of Privileges ... 690, 696, 697
Empfänger ... 60
enabled ... 242
Encoding ... 178
endAppearanceTranssition ... 390
Endlosrekursion ... 82, 136
endPoint ... 577

endRefreshing ... 779
endTrackingWithTouch:withEvent: ... 248
Entität ... 669
Entitätstyp ... 435
Entitlement ... 719
Entitlements ... 44
Entity-Relationship-Diagramm ... 434
Entwicklerzertifikat ... 43
Ereignis ... 238
Value Changed ... 237
Ergänzender View ... 514
Escape-Sequenz ... 286, 782
Event ... 236
Event-Handling ... 735
EXC_BAD_ACCESS ... 305
executeFetchRequest:error: ... 482, 495
Exit-Methode ... 339
expectedContentLength ... 830
Extensible Markup Langunge ... 786
Extreme Programming ... 975

F

Fast Enumeration ... 107, 187
Faustregel für Ressourcenverwaltung ... 267
Fehler ... 660
Fehlerbehandlung ... 985
Fehlerbericht ... 912, 913
Fehlersuche ... 292
fetchLimit ... 495
Fetchrequest ... 804
File Template Library ... 86, 888
File Transfer Protocol ... 821
File's Owner ... 107, 227, 265, 308
fillColor ... 579
Fingerposition ... 236
finish ... 300
Flowlayout ... 508
Flugmodus ... 865
Formatstring ... 181
forwardInvocation: ... 546
Foundation ... 174
Foundation-Framework ... 101, 117
und Core Foundation ... 597
Fowler, Martin ... 901
Frame ... 206
einer Animation ... 602
frame ... 413, 557
Framework
Core-Foundation- ... 781
CoreLocation- ... 873
Framework (Forts.)
MapKit- ... 869
SystemConfiguration- ... 865
TouchXML- ... 813
Fremdcode ... 981, 982
fromValue ... 592
FTP ... 821
Funktionstest ... 548
Fußleiste ... 344
Fuzzing ... 986

G

Gamma, Erich ... 169
Gang of Four ... 169
Garbage Collection ... 149
Garbage Collector ... 149
GCC ... 65
gdb ... 298
Geheimnisprinzip ... 59, 67, 79
Gemeinsame Oberklasse ... 906
Generic Passwords ... 719
genstrings ... 285
geocodeAddressDictionary:completion-
Handler: ... 875
Geocoder ... 873
Geografische Koordinate ... 871
Geokoordinaten ... 872
Geräteausrichtung ... 367, 416
Geräteidentifizierung ... 946
Geräteregistrierung ... 946
Geräteverwaltung ... 910
Gestenverarbeitung ... 258
Gesture Recognizer ... 255
GET-Anfrage ... 824, 850
Getter ... 60, 88
getter ... 154
Git ... 75, 952
Client ... 962
Server ... 967
git (Kommandozeilenprogramm) ... 965
gitolite ... 969
Gitter ... 507
GitX ... 962
Globale Suche ... 886
gl-setup ... 971
Glüheffekt ... 242
Goldberg, Adele ... 58
Go-Live-Test ... 988
Google ... 770, 869
Gradientenbutton ... 580

Grafikkontext ... 208
Grafische Benutzerschnittstelle ... 33
Greenwich Mean Time ... 217
Grenzmarke für POST-Anfragen ... 856
Größeninspektor ... 203, 205
Gruppe ... 202
GUI ... 33

H

Hardware-Tastatur ... 448
Hauptachsentransformation ... 611
Headerdatei ... 37, 79
importieren ... 100
Heap ... 136
Heise ... 655
Helm, Richard ... 169
help ... 299
hidden ... 567
highlighted ... 242
Hilfseditor ... 37, 228
Hintergrundfarbe ... 210
hitTest:withEvent: ... 246
Hochformat ... 410
HTTP ... 863
HTTP-Methode
GET ... 824
POST ... 850
Hybriddarstellung ... 870

I

IBAction ... 236, 340
IBOutlet ... 228
IBOutletCollection ... 232
iCloud ... 44
Icon ... 278
iControl ... 656
id ... 68, 69, 99
mit Protokoll ... 132
identifier ... 260
Identitätsinspektor ... 205
If-Else-Kette ... 805
imageEdgeInsets ... 243
ImageIO-Framework ... 596
imageNamed: ... 280, 627
Immutables ... 174
Implementierung ... 984
Implementierungsblock ... 40
Implementierungsdatei ... 37, 79
Implementierungsdetail ... 59, 79, 197
Implementierungsfehler ... 663
Implementierungsrichtlinie ... 985
Importanweisung ... 229, 231
Inaktive Applikation ... 129
indexPathForRowAtPoint: ... 476
Individual iOS Developer Program ... 23
Info.plist ... 286, 936
InfoPlist.strings ... 286
Information Disclosure ... 689, 695
Ingalls, Dan ... 58
In-House-Distribution ... 913
init ... 70, 83, 85, 206, 262
Initialer Viewcontroller ... 308, 351, 379
Initialisierungsmethode ... 85
initWithBytes:length: ... 184
initWithCoder: ... 220, 262, 480
initWithContentsOfFile:encoding:error: ... 177
initWithContentsOfURL: ... 822
initWithContentsOfURL:encoding:
error: ... 177, 823
initWithContentsOfURL:options:error: ... 822
initWithContentsOfURL:used-
Encoding:error: ... 823
initWithData:encoding: ... 769
initWithFrame: ... 205, 206, 220, 480, 513
initWithNibName:bundle: ... 264
und Storyboards ... 327
initWithStream: ... 838
initWithString: ... 207
Input-Accessory-View ... 448
inputAccessoryView ... 452
insertRowsAtIndexPaths:withRow-
Animation: ... 492
insertSections:withRowAnimation: ... 493
Installationshandbuch ... 987
Installationsprozess ... 988
instanceAlphaOffset ... 590
instanceCount ... 590
instancesRespondToSelector: ... 72, 941
instantiateViewControllerWith-
Identifier: ... 326, 385
Instanzvariable ... 66
Instruments ... 305, 640
Integrierte Entwicklungsumgebung ... 22
Integrität ... 690
Integritätsbedingung ... 442
Interface ... 79
Interface Builder ... 77, 889
interfaceOrientation ... 370, 416
Internet Passwords ... 719
Introspektion ... 71

iOS
Developer Program 22
Enterprise Program 23
Provisioning Portal 44
Simulator 897
University Program 23
iPad Simulator 290
IPA-Datei 914
iPhone 5 280, 288, 413, 424
iPhone-Konfigurationsprogramm 948
iPod touch 21
Is Initial View Controller 322
isEqual: 180
isEqualToString: 180
ISMS 977
ISO 27001 977
ISSECO 975
Issue Navigator 292, 886
Ist-Beziehung 63
isViewLoaded 266
iTunes Connect 912, 914
iTunes Store 22, 913
ivar 66

J

Jailbreak 660
Johnson, Ralph 169, 901
JSON 770
JSONKit 784

K

Kamera 25
Karten-App 878
Kartendarstellung 870
Kategorie 112, 249
Kategoriemethode 115
Kay, Alan 58
kCAAnimationDiscrete 598
kCAAnimationLinear 598
Keep it simple, stupid 697
Kennzahlen 986
Key Value Observing 542
Keychain 660, 718
Keyframe-Animation 595
Keypath 159
Keys 719
Key-Value-Coding 159
und Dictionary 187
Key-Value-Observing 74, 120, 361
Klasse 59
Klassenbasiert 59
Klassendeklaration 66
Klassenmethode 66, 85
und self 82
Klassenobjekt 66, 82
Klon 955
Konfiguration 920, 930
erben 939
Konflikt 959
Konsistenz 530, 531
Konsole 911
Konstruktor 70, 85
Kontextdiagramm 666
Koordinatensystem 871
Kopfleiste 344
Kryptografie 984
kSecAttrAccessibleAfterFirstUnlock 720
kSecAttrAccessibleAfterFirst-
UnlockThisDeviceOnly 720
kSecAttrAccessibleAlways 720
kSecAttrAccessibleAlwaysThis-
DeviceOnly 720
kSecAttrAccessibleWhenUnlocked 720
kSecAttrAccessibleWhen-
UnlockedThisDeviceOnly 720
KVC 159
KVO 120, 542

L

Label 34
Last-In-First-Out 544
Last-Modified 828
Laufzeitfehler 292, 294
Laufzeitumgebung 56
Launch-Image 287
Layer 559, 570
layerClass 571, 623
Layer-Klasse festlegen 571
layoutAttributesForDecoration-
ViewOfKind:decorationViewKind:
atIndexPath: 526
layoutAttributesForElementsInRect: 518
layoutAttributesForItemAtIndexPath: 518
layoutAttributesForSupplementary-
ViewOfKind:atIndexPath: 523
layoutIfNeeded 418
Layoutlogik 419
layoutSubviews 413, 417, 635
Lazy-Getter 276

Leak ... 138
Leaks ... 642
Least Privilege ... 698
length ... 184
levelsOfDetail ... 624
levelsOfDetailBias ... 624
libxml2 ... 811
LIFO ... 544
lineNumber ... 809
Linux ... 660
Lion ... 22
lldb ... 298
loadNibNamed:owner:options: ... 393, 475
loadView ... 266, 393
Localizable.strings ... 285
locationInView: ... 236, 257
Log Navigator ... 887
Logging ... 293
Login-Kurzname ... 861
Lokale Variable ... 66
Lokalisierung ... 25, 282
Lose Kopplung ... 120
Lower Camel Case ... 71

M

mainQueue ... 834
mainScreen ... 282
MainWindow.xib ... 261, 309
Mandatory Code Signing ... 43
Mapitem ... 878
mapItemForCurrentLocation ... 879
mapType ... 870
Mapview ... 870
mapView:annotationView:calloutAccessoryControlTapped: ... 877
mapView:viewForAnnotation: ... 876
masksToBounds ... 572
Maßnahme ... 661
Master ... 955
Master-Detail ... 363
Masterviewcontroller ... 362
maximumZoomScale ... 615
Medienbibliothek ... 889
Megginson, David ... 795
Mehreintrag ... 375
Mehrfachtap auswerten ... 241
Memory Mapping ... 184
Menge ... 187
Mercurial ... 963
Message Passing ... 58
Messgröße ... 989
Messinstrument ... 640
Methode ... 59
abstrakte ... 69
auslagern ... 904
Delegate- ... 126
erforderliche ... 117
optionale ... 117
überschreiben ... 64
Methodenimplementierung ... 40
Microsoft ... 665
SDL Pro Network ... 698
SDL Threat Modeling Tool ... 698
Threat Analysis & Modeling ... 703
MIME-Typ ... 825
minimumInteritemSpacing ... 509
minimumLineSpacing ... 509
minimumZoomScale ... 615
MKAnnotation ... 875
MKAnnotationView ... 876
MKCoordinateRegion ... 871
MKCoordinateSpan ... 871
MKLaunchOptionsDirectionsModeKey ... 880
MKMapItem ... 878
MKMapPointForCoordinate ... 872
MKMapRect ... 872
MKMapRectUnion ... 872
MKMapView ... 869
MKPinAnnotationView ... 876
MKPlacemark ... 878
MKUserLocation ... 877
Mobilfunk ... 25
Modaler Dialog ... 324
modalInPopover ... 330
modalTransitionStyle ... 328
modalViewController ... 325
Modell ... 73
Model-View-Controller ... 73
Modultest ... 548
Mountain Lion ... 22
Multipart-Datenformat ... 856
Multithreading ... 152
mutableCopy ... 176, 496
Mutables ... 174
MVC ... 73
Architekturmuster ... 248
Muster ... 307, 409

N

Nachricht ... 60, 64
Nachrichtenaustausch ... 58, 64
Namensraumdeklaration in XML ... 790
navigationBar ... 346
Navigationcontroller ... 333
und Autorotationsverhalten ... 415
navigationController ... 335
navigationItem ... 336
Navigationsleiste ... 330, 335
Navigatorspalte ... 885
nc ... 863
Need to know ... 697
needsDisplayForKey: ... 576, 602
netcat ... 863
next ... 300
NIB-Datei ... 308
Eigentümer ... 308
Nicht wiederherstellbare Ressource ... 266
nil ... 137, 277
NIST ... 666
nonatomic ... 154
Nonrepudiation ... 690
Notification ... 74, 130, 406
Notificationcenter ... 406
NSArray ... 160, 377, 822
NSArray-Literal ... 184
NSAutoreleasePool ... 150
NSBundle UIKitAdditions ... 475
NSCalendar ... 217
NSClassFromString ... 882, 941
NSCoder ... 220, 221
NSData ... 184, 769, 773, 821, 822
NSDate ... 84, 217, 271
NSDateComponents ... 271
NSDateFormatter ... 83, 217
NSDictionary ... 186, 542, 597, 822
NSDictionary-Literal ... 186
NSEntityDescription ... 442
NSFastEnumeration ... 853
NSFetchedResultsController ... 487
NSFetchedResultsControllerDelegate ... 489
NSFetchRequest ... 482
NSFileManager ... 841
NSFileProtectionComplete ... 732
NSFileProtectionCompleteUnlessOpen ... 732
NSFileProtectionCompleteUntilFirst-UserAuthentication ... 732
NSFileProtectionKey ... 732, 733
NSFileProtectionNone ... 732
NSIndexPath ... 473
NSInputStream ... 840
NSInteger ... 205
NSInvocation ... 544
NSJSONSerialization ... 772
NSKeyValueChangeOldKey ... 543
NSKeyValueCoding ... 159
NSKeyValueObserving ... 120
NSKeyValueObservingOptionOld ... 543
NSLayoutConstraint ... 425
NSLayoutFormatOptions ... 429
NSLayoutRelationEqual ... 426
NSLocalizedString ... 284
NSLog ... 82, 293, 911, 931
NSManagedObject ... 435
NSManagedObjectContext ... 442
NSMutableArray ... 92, 97, 185, 496
NSMutableData ... 836, 857
NSMutableRequest ... 851
NSMutableString ... 114, 156, 180, 799
NSMutableURLRequest ... 826
NSNotification ... 406
NSNotificationCenter ... 406
NSNull ... 603
NSNumber ... 95, 205, 277, 377, 542, 592
NSNumber-Literal ... 183, 185
NSObject ... 62, 85, 102, 142, 159
Protokoll ... 639
NSOperationQueue ... 834
NSOutputStream ... 839
NSPredicate ... 483
NSPropertyListSerialization ... 794
NSRange ... 844, 862
NSSet ... 160, 247, 437
NSSortDescriptor ... 483
NSString ... 68, 114, 156, 294
NSStringEncoding ... 781
NSStringFromSelector ... 82
NSTimeInterval ... 249
NSUInteger ... 542
NSUndoManager ... 544
NSURL ... 207, 597
NSURLAuthenticationChallenge ... 846
NSURLConnection ... 823, 833, 861
NSURLConnectionDataDelegate ... 835
NSURLConnectionDelegate ... 834
NSURLCredential ... 848
NSURLCredentialStorage ... 849
NSURLProtectionSpace ... 848
NSURLRequest ... 825
NSURLResponse ... 830

NSUserDefaults ... 355
NSValue ... 449, 592, 610
NSXMLParser ... 797
NSXMLParserDelegate ... 797
NULL ... 137
Nullmeridian ... 871
Nullobjekt ... 603
numberOfSectionsInCollectionView: ... 513
numberOfSectionsInTableView: ... 473
Nutzerrestriktion ... 421
Nygaard, Kristen ... 58

O

Oberklasse ... 61
objc_msgSend ... 650
objectAtIndex: ... 185
Objective-C ... 55
Objekt ... 58
  delegierendes ... 126
Objektbibliothek ... 34, 888
Objektgraph ... 137, 434
Objektkontext ... 441
Objektorientierte Programmierung ... 56
Objektvariable ... 66
observeValueForKeyPath:ofObject:change:
  context: ... 121, 124, 543
OCTAVE ... 665
OOP ... 56
opacity ... 572
Opaker Datentyp ... 208
Opdyke, William ... 901
openInMapsWithLaunchOptions: ... 879
openMapsWithItems:launchOptions: ... 880
openssl ... 46
openURL: ... 881
Operation-Queue ... 834
Organizer ... 293, 910, 960
orientation ... 416
orthogonal ... 255
Ortsmarke ... 878
Outlet ... 38, 227
  -Attribut ... 230
  -Collection ... 227, 231
  -Property ... 230
Outlet-Collection
  Speicherverwaltung ... 267

P

p ... 300
Pageviewcontroller ... 381
pageViewController:spineLocationFor-
  InterfaceOrientation: ... 381, 385
pageViewController:viewController-
  AfterViewController: ... 384
pageViewController:viewController-
  BeforeViewController: ... 384
PAL ... 690
Parameter ... 66
Parameterliste ... 70
parentViewController ... 328
parse ... 797
Parser ... 770, 809
  DOM ... 811
  JSON ... 772
  SAX ... 795, 810
  XML ... 788
parser:didEndElement:namespaceURI:
  qualifiedName: ... 805
parser:didStartElement:namespaceURI:
  qualifiedName:attributes: ... 805
parser:foundCharacters: ... 799
parser:parseErrorOccurred: ... 809
path ... 579, 593
PDCA ... 977
Pentesting ... 986
perfomSelector:withObject:afterDelay: ... 466
perform ... 333
Performance-Leak ... 652
performSegueWithIdentifier:sender: ... 259, 261
performSelector:withObject:
  und automatisches Referenzenzählen ... 806
performSelector:withObject:afterDelay: ... 837
performSelectorInBackground:with-
  Object: ... 152, 467
performSelectorOnMainThread:
  withObject:waitUntilDone: ... 468
Persistenz ... 434
Perspektivische Darstellung ... 609
PHP aktivieren ... 855
Pinchgeste ... 611
PKI-Umgebung ... 44
Plakette ... 374
Platzhalter ... 891, 893
plutil ... 794
po ... 300
pointInside:withEvent: ... 246, 251
Polling ... 120

pop ... 136
Pop-over ... 329, 360
Controller ... 456
Viewcontroller ... 364
popToRootViewControllerAnimated: ... 335
popToViewController:animated: ... 335
popViewControllerAnimated: ... 335
Port ... 863
position ... 593
POST-Anfrage ... 850
erzeugen ... 854
postNotification: ... 408
postNotificationName:object: ... 408
postNotificationName:object:userInfo: ... 408
Prädikat ... 482
Präsentations-Layer ... 584
prepareForSegue:sender: ... 260, 379
prepareLayout ... 519
prepareWithInvocationTarget: ... 546
presentingViewController ... 328
presentModalViewController:animated: ... 325
presentPopoverFromBarButtonItem:
permittedArrowDirections:animated: ... 330
presentPopoverFromRect:inView:
permittedArrowDirections:animated: ... 330
presentViewController:animated:
completion: ... 325, 386
previousFailureCount ... 846
print ... 300
Printf-Debugging ... 294
print-object ... 300
Private API ... 59
Produkt ... 202, 919
Profildatei ... 948
Profilinstallation ... 948
Programmbibliothek ... 922
Programmiersprache ... 55
Programmzeiger ... 295
Project Builder ... 883
Projekt ... 919
anlegen ... 29
ausführen ... 31
Projektdatei ... 913
Projektnavigator ... 885
Projektverwaltung ... 912
Property ... 88, 909
animierbar ... 602
bounds ... 206
frame ... 206
private ... 116
Read-Only-Eigenschaft ... 224
Property ... 88, 909 (Forts.)
redeklarieren ... 398
rootViewController ... 322
propertyListWithData:options:
format:error: ... 794
Protection class ... 720
Protokoll ... 117
informelles ... 572
Provisioning Profile ... 43
Proxy ... 400, 547
Proxymuster ... 547
Prozess ... 669, 975
Prozessgruppe ... 670
Prozessorarchitektur ... 916
Pseudozufallszahl ... 534
Punktnotation ... 90, 106, 148
und Debugger ... 300
push ... 136
pushViewController:animated: ... 334

Q

Qmail ... 990
QuartzCore-Framework ... 570
Quelle ... 669
Querformat ... 410

R

rand() ... 534
Range ... 840
rangeOfString: ... 179, 862
readonly ... 154, 224
readwrite ... 154, 224
Redo ... 442, 543, 545
Stack ... 545
Refactoring ... 901
Reference Counter ... 140
Referenzenzählen ... 139, 140
Referenzenzähler ... 142, 644
Referenzzeitpunkt ... 217
Reflexion ... 71
Refreshcontrol ... 778
refreshControl ... 779
region ... 872
registerClass:forCellReuseIdentifier: ... 480
registerClass:forCellWithReuseIdentifier: ... 513
registerClass:forSupplementaryView-
OfKind:withReuseIdentifier: ... 515
registerDefaults: ... 359
registerNib:forCellReuseIdentifier: ... 479

registerNib:forCellWithReuseIdentifier: 512
registerNib:forSupplementaryView-
OfKind:withReuseIdentifier: 515, 525
registerUndoWithTarget:selector:object: 544
Reiternavigation 371
Relationship 434
Many-To-Many 437
Many-To-One 437
One-To-Many 437
release 142, 467
Release-Konfiguration 930
reloadItemsAtIndexPaths: 526
removeAllAnimations 591
removeAnimationForKey: 591
removeAnnotation: 876
removeAnnotations: 876
removeCredential:forProtectionSpace: 849
removeFromParentViewController 388
removeFromSuperview 388
removeGestureRecognizer: 255
removeLastObject 97
removeObserver: 407
removeObserver:forKeyPath: 120
removeObserver:name:object: 407
removeOnCompletion 591
removeTarget:action: 256
renderInContext: 583
repeatCount 591
repeatDuration 591
replaceCharactersInRange:withString: 181
Replication-Layer 589
Repudiation 689, 694
Request 774
requestWithURL: 825
requestWithURL:cachePolicy:time-
outInterval: 825
Requirement 981
reset 256
Responder 247
respondsToSelector: 71, 341, 397, 941
Response-Policy 988
Ressource 266
Ressourcenverzeichnis 471
Restriktion 419
retain 142, 154
Retain-Zyklus 643
Retina-Display 280
RFC 3986 783
RGBA-Wert 209
Richtlinien 983
Richtungsvektor 609
Root Certificate Authority 48
Root-CA 48
Rootviewcontroller
- des Navigationcontrollers 333
App 308
rootViewController 334, 379, 391
Rotation
an Hauptachsen 610
Rotationsachse 609
Rotationsmatrix 562
row 473
Rückruffunktion 194
runActionForKey:object:arguments: 603
Runloop 150, 868
RUP 975

S

Sandbox 470, 911
Satellitenansicht 870
SAX 795
SAX-Ereignis 795
Characters 799
End Document 810
End Element 801, 802, 808
Start Document 810
Start Element 802, 805
scale 281, 557, 583
scheduledLocalNotifications 270
scheduledTimerWithTimeInterval:target:
selector:userInfo:repeats: 224
scheduleLocalNotification: 270
Schema 941
Schema-Auswahl 897
Schicht 73
Schlüssel 186, 807
Schlüsselbund 660
Schlüsseleigenschaft 807
Schlüsselmanagement 988
Schnellhilfe 918
Schnittstelle 79
Schwache Referenz 165
und iOS 4 168
Schwachstellenanalyse 987
SCNetworkReachabilityFlags 867
SCNetworkReachabilityGetFlags 865
SCNetworkReachabilityRef 867
SCNetworkReachabilitySetCallback 868
SCNetworkReachabiltyCreateWithName 867
Screenshot 582
scrollToPoint: 579

scrollToRect: 579
Scrollview 611
scrollViewDidEndDragging: 776
scrollViewDidScroll: 776
Scrum 975
SDK 22
SDL 975
searchDisplayController:shouldReload-
TableForSearchString: 494
section 473
Secure by Default 698
Secure Coding Guide 656
Security Advisor 981
Security Policy 977
Security Response 988
Security-Datenbank 981
Security-Expertise 982
Security-Testing 986
Security-Training 980
Segmented Control 353
Segue 227, 259
selected 242
Selektor 69
self 70, 82, 207
sendAsynchronousRequest:queue:
completionHandler: 834
Sendmail 990
sendSynchronousRequest:returning-
Response:error: 823
Senke 669
SenTestCase 549
SenTestKit 550
Sequenzdiagramm 94
Serialisierung 221, 765
binäre 766
setBackgroundImage:forBarMetrics: 347
setBackgroundImage:forToolbarPosition:
barMetrics: 347
setCenterCoordinate:animated: 871
setDateFormat: 83
setFinishedSelectedImage:withFinished-
UnselectedImage: 381
setHTTPBody: 826
setHTTPMethod: 826
setNeedsDisplay 209, 358, 584
setNeedsLayout 413, 418
setPrimitiveValue:forKey: 447
setRegion:animated: 872
setSelectedItem: 127
Setter 60, 88
setter 153
setText: 83
setTranslatesAutoresizingMaskInto-
Constraints: 427
setUp 549
setValue:forHTTPHeaderField: 826
setValue:forKey: 159, 574
setViewControllers:direction:animated:
completion: 384
setVisibleRect:animated: 873
sharedApplication 128
shouldAutomaticallyForward-
AppearanceMethods 390
shouldAutomaticallyForward-
RotationMethods 390
shouldAutorotate 368, 415
shouldAutorotateToInterface-
Orientation: 367, 414
shouldProcessNamespaces 797
shouldReportNamespacePrefixes 798
Sicherer Entwicklungszyklus 975
Sicherheit der Ausstattung 978
Sicherheitsanforderungen 982
Sicherheitskonzept 981
Sicherheitslücke 661
Sicherheitsmetrik 989
Sicherheitsüberprüfung 981
SIGABRT 304
Simple API for XML 795
Simula 58
Simulated Metrics 344
Singleton 128, 168
und Referenzenzähler 142
size 280, 557
sizeThatFits: 511
sizeWithFont: 349
Skalierungsfaktor 281
Smalltalk-80 58
Snapshot 172, 898
automatischer 900, 903
exportieren 913
Snippet 90
Software Development Kit 22
Software-Tastatur 448
sortedArrayUsingSelector: 185
soundName 271
SourceTree 963
Spätes Binden 65
speed 591
Speicher 135
Speicherleck 138, 638, 642
Speicherverwaltung 55, 139

Speicherverwaltungsregeln 143, 645
und Outlet-Attribute 230
Speicherzugriffsfehler 638
Spiegelung 561, 582
Spiralmodell 975
Splitviewcontroller 362
splitViewController 367
splitViewController:popoverController:
willPresentViewController: 363
splitViewController:willHideView-
Controller:withBarButtonItem:for-
PopoverController: 363
386splitViewController:willShowViewCon-
troller:invalidatingBarButtonItem: 363
Spoofing 689, 693
Springboard 278
srand() 534
Stack 136, 800
Stackframe 136
Stack-Overflow 136
Stacktrace 305
Standardrestriktion 420, 421
Standardsprache 283
standardUserDefaults 355
Standortanzeige 870
Stapel 136, 800
Stapelüberlauf 136
startAnimating 465
Startbildschirm 287
startPoint 577
state 242, 258
Statisches Binden 65
Statusleiste 344
step 300
Stepper 98
stopAnimating 465
Storyboard 259, 316, 372, 478
storyboard 326, 385
strcpy 663
STRIDE 689
stringByAddingPercentEscapes-
UsingEncoding: 781
stringFromDate: 84
stringWithContentsOfURL:encoding:
error: 42, 822
stringWithContentsOfURL:usedEncoding:
error: 823
stringWithFormat: 83, 858
strokeColor 579
strong 154
Subklasse 61
sublayers 571
sublayerTransform 609
Subskriptionsoperator 185, 187
substringFromIndex: 862
substringToIndex: 862
substringWithRange: 179
Subtyp 61, 104
subtype 594
Subversion 75, 952
Subview 392
SubviewController 386, 392
subviews 233
Sun Tzu 662
super 82, 105
Superklasse 61
direkte 63
superlayer 571
Superview 206
supportedInterfaceOrientations 368, 415
SVN 75
Symbolnavigator 885
synchronize 356

T

tabBar:didSelectItem: 127
tabBarController:didSelectViewCon-
troller: 374
tabBarController:shouldSelectView-
Controller: 374
tabBarController:willEndCustomizingView-
Controllers:changed: 377
tabBarItem 373
Tabbarnavigation 371
Tableview 472
statisch 352
tableView:cellForRowAt-
IndexPath: 473, 478, 876
tableView:commitEditingStyle:for-
RowAtIndexPath: 481
tableView:numberOfRowsInSection: 473
Tag 476
tag 377
Tampering 689, 694
tapCount 240
Target 202, 548, 920, 934
Target Membership 549
target stop-hook list 303
Target-Action 236
Mechanismus 74, 248
Target-Einstellungen 288

Tastatur ... 447
Tastaturkurzbefehle ... 894
Tastenbelegung ... 894
Tastenkürzel ... 894
Team Provisioning Profile ... 50
tearDown ... 549
Temporäre Datei ... 470
Testdaten ... 979
Testgetriebene Softwareentwicklung ... 555
Testmethode ... 553
Test-Methodik ... 987
Testobjekt ... 553
Testtarget anlegen ... 548
textViewDidEndEditing: ... 448
textViewShouldBeginEditing: ... 448
Thread ... 152
Threat Modeling ... 664, 980
tileSize ... 626
Time Machine ... 953
Time Profiler ... 650, 653
timeOffset ... 600
Timer ... 223
tintColor ... 337
titleEdgeInsets ... 243
Toll-Free Bridge ... 637, 782
Toll-Free Bridging ... 597
TomTom ... 869
toolbarItems ... 343
Tools ... 980
Top-Level-Objekt ... 235
Speicherverwaltung ... 267
touchesBegan:withEvent: ... 247
touchesCancelled:withEvent: ... 247, 618
touchesEnded:withEvent: ... 247
touchesMoved:withEvent: ... 247
touchesShouldBegin:withEvent:
inContentView: ... 617
touchesShouldCancelInContentView: ... 618
TouchXML ... 811
toValue ... 592
Training ... 980
Trainingsplan ... 980
transform ... 560, 608
Transformation
eines Layers ... 608
eines Views ... 560
Transformationsmatrix ... 560, 589
Transformationsmatrix ... 588
Transition ... 565
transitionWithView:duration:options:
animations:completion: ... 566
translatesAutoresizingMaskInto-
Constraints ... 427
Trust Boundary ... 672
Trust-Modelle ... 983
TU Wien ... 658
Twitter ... 770
TWTweetComposeViewController ... 500, 504
Typisierung ... 67
Typlosigkeit ... 69

U

Übergang ... 259, 324, 332
Überschreiben ... 61, 64
UDID ... 658, 946
UI_APPEARANCE_SELECTOR ... 403
UIAcceleration ... 538
UIAccelerometer ... 538
UIActivityIndicator ... 465
UIAlertView ... 274, 846
UIApplication ... 126, 270
UIApplicationDelegate ... 128
UIApplicationMain ... 308, 322
UIApplicationStateActive ... 274
UIBarButtonItem ... 337, 343, 344
UIBarButtonSystemItemCompose ... 351
UIBarButtonSystemItemDone ... 337
UIBarButtonSystemItemSave ... 337
UIButton ... 582
UIButtonTypeCustom ... 243
UICollectionElementKindSectionFooter ... 514
UICollectionElementKindSectionHeader ... 514
UICollectionReusableView ... 514
UICollectionViewCell ... 514
UICollectionViewController ... 507
UICollectionViewDelegateFlowLayout ... 511
UICollectionViewFlowLayout ... 509, 517
UICollectionViewLayout ... 517
UICollectionViewLayoutAttributes ... 516
UIControl ... 236, 242, 248, 618
UIControlEventEditingDidBegin ... 241
UIControlEventEditingDidChange ... 241
UIControlEventEditingDidEnd ... 242
UIControlEventEditingDidEndOnExit ... 242
UIControlEventTouchCancel ... 240
UIControlEventTouchDown ... 240
UIControlEventTouchDownRepeat ... 240
UIControlEventTouchDragEnter ... 240
UIControlEventTouchDragExit ... 240
UIControlEventTouchDragInside ... 240
UIControlEventTouchDragOutside ... 240

UIControlEventTouchUpInside 240
UIControlEventTouchUpOutside 240
UIControlEventValueChanged 239
UIControlStateDisabled 242
UIControlStateHighlighted 242
UIControlStateSelected 242
UIDatePicker 239
UIDevice 291, 313, 416
UIEdgeInsetsMake 612
UIEvent 247
UIGestureRecognizer 255, 256
UIGestureRecognizerStateEnded 258
UIGraphicsBeginImageContext 461
UIGraphicsBeginImageContextWith-
Options 583
UIGraphicsGetCurrentContext 209, 461
UIImage 280, 557, 572
und Caching 627
UIImageJPEGRepresentation 458
UIImagePickerController 329, 455
UIImagePNGRepresentation 458
UIInterfaceOrientationIsLandscape() 368
UIInterfaceOrientationIsPortrait() 368
UIInterfaceOrientationLandscapeLeft 368
UIInterfaceOrientationLandscapeRight 368
UIInterfaceOrientationMaskAll 369
UIInterfaceOrientationPortrait 368
UIInterfaceOrientationPortrait-
UpsideDown 368
UIKeyboardDidShowNotification 448
UIKeyboardWillHideNotification 448
UIKeyboardWillShowNotification 448
UIKit 100
UIKit-Framework 100
UILocalNotification 269, 409
UILocalNotificationDefaultSoundName 271
UILongPressGestureRecognizer 256
UIModalTransitionStyleCoverVertical 328
UIModalTransitionStyleCrossDissolve 328
UIModalTransitionStyleFlipHorizontal 328
UIModalTransitionStylePartialCurl 328
UINavigationBar 335, 345
UINavigationController 415
UINavigationItem 336
UINib 479
UIPageControl 239
UIPageViewController 381
UIPageViewControllerDataSource 384
UIPageViewControllerDelegate 385
UIPageViewControllerOptionSpine-
LocationKey 384
UIPageViewControllerSpineLocationMax .. 381
UIPageViewControllerSpineLocationMid ... 381
UIPageViewControllerSpineLocationMin ... 381
UIPageViewControllerTransition-
StylePageCurl 383
UIPageViewControllerTransition-
StyleScroll 383
UIPanGestureRecognizer 255, 617
UIPinchGestureRecognizer 255
UIPopoverArrowDirectionAny 331
UIPopoverController 330
UIRefreshControl 239, 777
UIRequiredDeviceCapabilities 916
UIResponder 247, 256
UIRotationGestureRecognizer 255
UIScreen 282
UIScrollView 579, 612, 776
UIScrollViewDelegate 615
UISearchBar 493
UISearchBarDelegate 780
UISearchDisplayController 493
UISegmentedControl 239
UISlider 239
UISplitViewController 363
UIStepper 98, 239
UIStoryboard 326
UIStoryboardSegue 333, 340
UISwipeGestureRecognizer 255
UISwitch 237, 239, 353
UITabBar 127, 374
UITabBarController 372
UITabBarControllerDelegate 379
UITabBarDelegate 127
UITabBarItem 373
UITableView 472
UITableViewCell 474
UITableViewController 354
UITableViewDataSource 472
UITableViewDelegate 472
UITapGestureRecognizer 255, 616
UITextField 241, 447
UITextView 447
UIToolBar 452
UIToolbar 345, 346, 451
UITouch 236, 247
UIUserInterfaceIdiomPad 291, 313
UIUserInterfaceIdiomPhone 291, 313
UIView 201, 247, 413
UIViewAnimationOptionAllow-
AnimatedContent 568
UIViewController 130, 227, 247, 262

UIWindow 309, 391, 392
Umgebender View 206
Umgebung 978
UML 59
Umlaute 766
Unabhängige Viewklasse 225
undisplay 303
Undo 442, 543
Manager 544
Stack 544
Ungeordnete Menge 187
Unicode 767
Unit-Test 548
ausführen 551
Universelle App 313
Unterklasse 61
Unveränderliche Objekte 174
Unwind-Übergang 339
Upper Camel Case 71
URI 790
URL
Anfrage 774
Anfrage, synchrone 774
Kodierung 782
Protokoll 821
useCredential:forAuthentica-
tionChallenge: 846
User-Agent 825
Userdefaults 357, 376
userInteractionEnabled 246, 618
userInterfaceIdiom 291, 313
Utilities-Ansicht 34
Utilitybereich 887

V

valueForKey: 159, 187, 574
valueForKeyPath: 377, 773
values 593, 597
valueWithCATransform3D: 610
Veillard, Daniel 811
Veränderliche Objekte 174
Verbindung ziehen 228
Verbindungsinspektor 123, 230
Vererbung 58, 61
Verfügbarkeit 690
Vergleichsoperator 484
Verhalten 58, 104
Versionsverwaltung 75, 898, 979
Vertraulichkeit 690
view 262, 266
Viewausrichtung 367, 416
Viewcontroller 226
Anzeigezyklus 266
Lebenszyklus 262
viewControllers 363
viewDidAppear: 227, 262, 266
viewDidDisappear: 130, 227, 266
viewDidLayoutSubviews 389, 419
viewDidLoad 81, 245, 266
viewDidUnload 229, 264, 267
viewForZoomingInScrollView: 615
Viewgeometrie 206
Viewhierarchie 235
Viewtransition 565
viewWillAppear: 54, 227, 266, 361, 388
viewWillDisappear: 130, 227, 266
viewWillLayoutSubviews 419
viewWillUnload 264
viewWithTag: 477
Visual Format Language 427
Vlissides, John 169
V-Modell 975
void 82
Vorabkontrolle 982
Vorwärtsdeklaration 119, 229
für ein Protokoll 396
für eine Klasse 397
Vorzeichenlose/-behaftete Darstellung 766

W

Wackelanimation 564
Warnung 292
Warteschlange 270
weak 154
und Outlets 354
WebDAV 821
Webfreigabe aktivieren 828
Werkzeugleiste 330, 343, 451, 884
Wiederherstellbare Ressource 266
Wiederverwendbarkeit 61
Wight, Jonathan 811
Wildcard-Profil 50
willChangeValueForKey: 125
willMoveToParentViewController: 388
willRotateToInterfaceOrientation: 369
willRotateToInterfaceOrientation:
duration: 416
window 317
Wireless Wide Area Network 866
Wischnavigation 381

Workspace ... 885, 920
wrapped ... 578
Wrapper-Bibliothek ... 811
writeToFile:atomically: ... 783, 794
writeToFile:atomically:encoding:error: ... 177
Wurzelklasse ... 62
WWAN ... 866
WWDR intermediate certificate ... 48
WWW-Konsortium ... 786, 811

X

Xcode ... 22, 883
Version 4.2 ... 255
XIB-Datei ... 290, 308
XML ... 786
-Deklaration ... 786
Entität ... 788
Namensraum ... 789
Path Language ... 814
Wurzelelement ... 786
XPath ... 814
Achsen ... 817
Prädikat ... 817

Y

YouTube ... 770

Z

Zeichenkette ... 68
Zeichenkettenkonstante
und Referenzenzähler ... 142
Zeichenkodierung ... 178, 766
ASCII ... 766
ISO Latin 1 ... 767
ISO-8859-1 ... 767
MacRoman ... 767
UTF-8 ... 768
Zeiger ... 137
Zeilennummern ... 890
Zeitachse ... 599
Zeitraum ... 599
Zeitverbrauch ... 650
Zertifikat ... 43
zIndex ... 516
Zombie ... 647
zPosition ... 608
Zugriffsmethode ... 60
Zurückbutton ... 336, 338
Zustand ... 58
Zyklische Abhängigkeit ... 118, 132
Zyklische Referenz ... 119